2002-06

Specifying quality of service for distributed systems based upon behavior models

Drummond, John.

Monterey, California: Naval Postgraduate School, 2002.

http://hdl.handle.net/10945/9778

NAVAL POSTGRADUATE SCHOOL
Monterey, California

DISSERTATION

SPECIFYING QUALITY OF SERVICE FOR DISTRIBUTED SYSTEMS BASED UPON BEHAVIOR MODELS

by

John Drummond

June 2002

Dissertation Supervisor: Valdis Berzins

Approved for public release; distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** June 2002	**3. REPORT TYPE AND DATES COVERED** Dissertation	
4. TITLE AND SUBTITLE: Title (Mix case letters) **Specifying Quality Of Service For Distributed Systems Based Upon Behavior Models**			**5. FUNDING NUMBERS**
6. AUTHOR(S) John Drummond			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			**12b. DISTRIBUTION CODE**

13. ABSTRACT ***(maximum 200 words)***

The substantial complexity and strict requirements of distributed command & control systems creates an environment that places extreme demands upon system resources. Furthermore, inconsistent resource distribution also introduces the distinct possibility of potential errors, and process failures. Many of these potential difficulties can be understood and addressed through a practical analysis of the resource management and distribution procedures employed within these systems. This analysis should include a direct focus upon the essential quality of service that is shared among the software programs that operate within this environment. However, the current approaches to this analysis are lacking in that there is no accurate method to determine precisely what quality of service based conflicts take place during program execution. This problem can be addressed through examination of specific quality of service actions during program execution. To achieve a precise analysis of quality of service actions this dissertation research has implemented an approach to examine the exact quality of service execution path during program operation.

14. SUBJECT TERMS Event Trace, Behavioral Model, Quality of Service, Command & Control			**15. NUMBER OF PAGES** 261
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited.

SPECIFYING QUALITY OF SERVICE FOR DISTRIBUTED SYSTEMS BASED UPON BEHAVIOR MODELS

John J, Drummond
B.S., San Diego State University, 1992
M.S., Naval Postgraduate School, 1997

Submitted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY IN SOFTWARE ENGINEERING
from the

NAVAL POSTGRADUATE SCHOOL
June 2002

Author: _____
John J, Drummond

Approved by:

_____
Dr. Valdis Berzins
Professor of Computer Science
Dissertation Supervisor

_____	_____
Dr. Luqi Professor of Computer Science	Dr. William Kemple Professor of Information Warfare
_____	_____
Dr. Mikhail Auguston Professor of Computer Science	Dr. Nabendu Chaki Professor of Computer Science

Approved by: _____
Luqi, Chair, Department of Software Engineering

Approved by: _____
Carson K. Eoyang, Associate Provost for Academic Affairs

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The substantial complexity and strict requirements of distributed command & control systems creates an environment that places extreme demands upon system resources. Furthermore, inconsistent resource distribution also introduces the distinct possibility of potential errors, and process failures. Many of these potential difficulties can be understood and addressed through a practical analysis of the resource management and distribution procedures employed within these systems. This analysis should include a direct focus upon the essential quality of service that is shared among the software programs that operate within this environment. However, the current approaches to this analysis are lacking in that there is no accurate method to determine precisely what quality of service based conflicts take place during program execution. This problem can be addressed through examination of specific quality of service actions during program execution. To achieve a precise analysis of quality of service actions this dissertation research has implemented an approach to examine the exact quality of service execution path during program operation.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I. INTRODUCTION 1
- **A. DOCUMENT LAYOUT 2**
- **B. GOALS AND NEW CONTRIBUTION 5**
- **C. PROBLEM INTRODUCTION, SPECIFIC GOALS 6**
- **D. PROBLEM SIGNIFICANCE, POTENTIAL IMPACT 16**

II. OVERVIEW 21
- **A. RESEARCH STRATEGY AND APPROACH 21**
- **B. TACTICS FOR PRODUCING NEW CONTRIBUTION 22**

III. PREVIOUS WORK 37
- **A. ASSESSMENT OF WORK 37**
- **B. RESULTS SUMMARY 38**

IV. EVENT TRACE ANALYSIS 53
- **A. TARGET PROGRAM 53**
- **B. OBJECTIVES OF THE EVENT TRACE ANALYSIS 54**
- **C. EVENT TRACE GRANULARITY 55**

V. TESTBED ENVIRONMENT 61
- **A. LINUX 63**
- **B. LINUX/RK 67**
- **C. ENSEMBLE 71**
- **D. FAST FAILURE DETECTOR 73**

VI. INVESTIGATION RESULTS 77
- **A. QUALITY OF SERVICE EVENT TYPES 77**
- **B. QUALITY OF SERVICE EVENT TRACE 81**

VII. CONCLUSION 89
- **A. ORIGINAL OBJECTIVES 89**
- **B. RESULTING FINDINGS 90**
- **C. PENDING ISSUES 101**
- **D. FUTURE WORK 102**

APPENDIX 105
- **A. QOS EVENT TRACE DATA STRUCTURE DOCUMENTATION 105**
  - **css_disk_cpu_reserve_attr Struct Reference 105**
  - **disk_param Struct Reference 106**
  - **et_qos_error_struct Struct Reference 107**
  - **et_sys_res Struct Reference 108**
  - **et_sys_task Struct Reference 109**
  - **et_system_status_struct Struct Reference 110**
  - **event_trace_struct Struct Reference 111**
  - **itimerspec Struct Reference 112**
  - **posix_timer Struct Reference 113**

B. QOS EVENT TRACE FILE DOCUMENTATION 114
8254.c File Reference 114
censustaker.c File Reference 116
censustaker.h File Reference 117
cpu_reserve.c File Reference 118
css.h File Reference 122
css_disk_cpu.c File Reference 124
disk_reserve.c File Reference 126
division.c File Reference 130
event_trace.c File Reference 132
event_trace.h File Reference 145
event_trace_system.c File Reference 151
event_trace_system.h File Reference 152
ffd.c File Reference 153
ffd.h File Reference 155
hbeatstats.c File Reference 156
heartbeat.c File Reference 157
heartbeat.h File Reference 158
hosts.c File Reference 159
hosts.h File Reference 160
iterator.c File Reference 161
longlong.h File Reference 162
messages.c File Reference 164
messages.h File Reference 165
misc.c File Reference 166
multicast.c File Reference 167
multicast.h File Reference 168
mutex.c File Reference 169
net_reserve.c File Reference 172
notify.c File Reference 173
notify.h File Reference 174
params.c File Reference 175
params.h File Reference 178
posix_timers.h File Reference 179
ppc_timer.c File Reference 180
reserve.c File Reference 182
resource_set.c File Reference 184
rk.h File Reference 187
rk_error.h File Reference 188
rk_init.c File Reference 189
rk_isr.c File Reference 191
rk_linux.h File Reference 193
rk_procfs.c File Reference 194
rk_sched.c File Reference 196
rt_process.c File Reference 197
timer.c File Reference 198

timespec.h File Reference **201**
urgency.c File Reference **204**
urgency.h File Reference **209**
utils.c File Reference **211**
utils.h File Reference **212**
C. COPYRIGHT AND LICENSE **212**
1. Fast Failure Detector **212**
2. Linux/RK **213**
3. Linux **214**
5. Ensemble **223**
DEFINITIONS, SYMBOLS, AND ACRONYMS **227**
LIST OF REFERENCES **231**
INITIAL DISTRIBUTION LIST **241**

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

FIGURE 1. LEVELS OF COMMAND 8
FIGURE 2. RESOURCE MANAGEMENT 17
FIGURE 3. RESOURCE MANAGEMENT SCOPE 18
FIGURE 4. QUALITY OF SERVICE ANALYSIS FRAMEWORK 23
FIGURE 5. QUALITY OF SERVICE PROGRAM POINTS 25
FIGURE 6. RESOURCE REQUEST EVENT MODEL 27
FIGURE 7. QUALITY OF SERVICE VIOLATION EVENT MODEL 27
FIGURE 8. QUALITY OF SERVICE LEVEL EVENT MODEL 28
FIGURE 9. RESOURCE NEGOTIATION EVENT MODEL. 29
FIGURE 10. SYSTEM RESERVATION EVENT MODEL 29
FIGURE 11. RESOURCE ASSIGN EVENT MODEL 30
FIGURE 12. PATH LENGTH EVENT MODEL 30
FIGURE 13. RESOURCE RE-NEGOTIATION EVENT MODEL 31
FIGURE 14. QUALITY OF SERVICE BEHAVIOR MODEL 32
FIGURE 15. QUALITY OF SERVICE SPECIFICATION TAXONOMY 50
FIGURE 16. EVENT TRACE SEQUENCE 58
FIGURE 17. DOMAINS WITHIN THE SSC-SD DARPA TESTBED 61
FIGURE 18. FAST FAILURE DETECTION TEST ENVIRONMENT 63
FIGURE 19. THE LINUX KERNEL 66
FIGURE 20. LINUX RESOURCE KERNEL ARCHITECTURE 69
FIGURE 21. RESOURCE RESERVATION PARAMETERS. 70
FIGURE 22. LAYERED MICROPROTOCOLS IN ENSEMBLE. 72
FIGURE 23. FAST FAILURE DETECTION PROGRAM 74
FIGURE 24. APPLYING THE EVENT MODEL 91
FIGURE 25. EMPLOYING THE QUALITY OF SERVICE ANALYSIS FRAMEWORK 94
FIGURE 26. QUALITY OF SERVICE ANALYSIS. 97
FIGURE 27. LEGACY PROGRAM IMPLEMENTATION 99
FIGURE 28. QUALITY OF SERVICE RESOURCE MANAGEMENT 100

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

TABLE 1. LIFECYCLE MODELS AND INTERACTION MODELS OF SIX OO METHODS 45
TABLE 2. QUALITY OF SERVICE EVENTS 77
TABLE 3. EVENT TRACE DATA-1 81
TABLE 4. EVENT TRACE DATA-2 84

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to acknowledge the invaluable assistance of the Ph.D. committee members Dr. Valdis Berzins (Chair), Dr. Luqi, Dr. William Kemple, Dr. Mikhail Auguston, and Dr. Nabendu Chaki. Their encouragement and beneficial advice have made this work possible. I would also like to acknowledge the DARPA Quorum program and the research and experiments performed within the Quorum Integration, Testbed and Exploitation (Quite) project. This DARPA[1] Quorum project effort is structured as follows: DARPA-ITO –Sponsor, SPAWAR –Technical/Contracting Lead, Teknowledge –Integration Lead, and System/ Technology Development Corporation (S/TDC) – Integration, The Open Group (TOG) Integration. The individuals within the Quite project include: Gary Koob Quorum Program Manager (DARPA-ITO), John Drummond, Al Sandlin (SPAWAR); Neil Jacobstein, Adam Pease, Lou Coker, Jeff Vu, Joe Marclino, Jim Reynolds (Teknowledge); Mustafizur Rahman, Jim Carroll (TOG); Arthur Robinson, Manoj Srivastava, Amarendranath Vadlamudi, Shivakumar Patil (S/TDC).

[1] This effort is funded by theDefense Advanced Research Projects Agency – Information Technology Office(DARPA-ITO)

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

This chapter presents an introduction to the thesis problem to be solved. The main objective and intention of this work is discussed here as well as the general and specific goals that have been pursued by this research. Relevant previous work within this area is also briefly reviewed, with the more detailed analysis of prior efforts deferred to chapter III. An outline of the overall layout of this document is also provided within this introductory chapter. The document layout section includes a brief reference to the focus of the individual chapters.

The object of this work is to directly address the problem of effectively evaluating quality of service efficiency for programs within the distributed command & control environment. From a software engineering perspective the completion of this objective will provide essential support for correctly characterizing a software engineering approach for distributed systems with respect to efficient employment of system resources.

The contributions to the software engineering state of the art include the development of event models and subsequent behavioral models (based upon quality of service actions) that can be applied to software development. Another contribution to the software engineering state of the art includes the creation of a framework for applying a quality of service behavioral analysis to software applications within the distributed command & control environment with respect to efficient employment of system resources. Additionally this work produces a contribution to the software engineering state of the art that includes the creation of a method for performing quality of service fine-tuning tasks upon legacy systems that contain mission critical applications that exhibit resource inefficiencies.

A. DOCUMENT LAYOUT

The layout of this document includes chapters as outlined below. These chapters may also contain sections as required to further describe this research effort. The focus of these chapters is briefly suggested here.

Introduction

This chapter presents an introduction to the thesis problem to be solved. The main objective and intention of this work is furnished here. A review of previous work within this area is also briefly touched upon. This chapter includes sections on goals and new contributions, problem introduction –specific goals, and problem significance – potential impact.

Overview

A synopsis of the development of behavioral models for distributed command & control system applications and the basic approach to be followed are presented in this chapter. The sections included within this chapter encompass the focus areas of research strategy and approach, and tactics for producing new contribution.

Previous Work

A detailed assessment of previous work within this focus of this research is discussed at length. The specific sections that comprise this chapter include assessment of work, and results summary.

Event Trace Analysis

The main techniques utilized for the quality of service event trace analysis work are discussed in this chapter. The sections within this chapter are target program, and event trace granularity.

Testbed Environment

The research environment that the quality of service event trace analysis effort has utilized is presented in this chapter. This includes descriptions of operating system details as well as application programs and testbed hardware. The testbed environment chapter is delineated with sections that include linux, linux/rk, ensemble, and fast failure detector.

Investigation Results

The resulting examination data and information is presented in this chapter including theoretical analysis of performance. The chapter sections include quality of service event types, and quality of service event trace.

Conclusion

This chapter presents a summary of the dissertation work that has been completed. Concluding elements contain restatement of the initial research work and the resulting findings. Pending issues from this work to as related to command & control, distributed computing, and collaborative processing are also discussed. The scope for future work is also discussed in this chapter. The sections contained within the conclusion chapter include original objectives, resulting findings, pending issues, and future work.

Appendix-A Data Structures

This chapter provides the software data structures that have been utilized for this research work. This documentation has been processed through the Doxygen source documentation application program.

Appendix-B Source Code Files

This chapter provides the source code files that have been utilized for this research work. This documentation has been processed through the Doxygen source documentation application program.

Appendix-C Copyright and License

This chapter includes copyright and license information as related to the source code utilized within this dissertation research.

Definitions, Symbols, and Acronyms

Specific acronyms and symbols utilized in this document are listed here in addition to definitions for jargon and complex terminology needing further description.

List of References

The works of researchers referenced in this document, and those mentioned for further reading are listed in this chapter.

B. GOALS AND NEW CONTRIBUTION

The long-term contributory goal of this work is to aid in the development of an approach for designing distributed command & control systems with regard to efficient resource utilization. The effort to achieve this goal entails the creation of a framework of a system and processes that coalesce the results of a quality of service based event trace performed within a targeted distributed system and the specific resource utilization/management needs of applications within this distributed system. This event trace[Auguston 99] is performed to verify quality of service efficiency levels through behavioral models. As stated in [Adriole 86] verification in software engineering performs a specific function, he defines verification as "…checking coherence between input and output products for a specific stage in the development process as opposed to validation which deals with the assessment of final products against initial requirements." In the context of the quality of service event trace a check for coherence between the request for resources and the granting of these resources will allow for such verification to be performed. The result of this effort is to produce a series of verifiable quality of service characteristics that can be applied to a generalized system design. Specific metrics have been defined for this quality of service analysis and are described in the next subsection.

The terms that describe the prominent individual elements of this work (e.g. command & control, distributed systems, quality of service), have assumed diverse meanings within the research community. Therefore while proceeding into a description of the problem that is the focus of this research, the specific definitions of these elements within the context of this research, are also provided at various segments in the following subsection of this document.

The distinct new contribution that this quality of service event trace research adds to the current software engineering domain is the development of a conceptual method

for supporting the inclusion of objectively measurable quality of service and resource management requirements into software engineering design. The comprehensive contribution to be produced by this research work is essentially a framework which consists of this method as well as processes that combine the results of a quality of service based event trace.

This framework can be readily applied to the analysis process that is performed upon typical distributed system and command & control environments. Additionally this work produces a set of characteristics, which can be utilized to assist in overall efficient quality of service systems and resource management element design. Resultant software and application of precise evaluation techniques for isolating quality of service specific uncertainties has also been produced by this work.

C. PROBLEM INTRODUCTION, SPECIFIC GOALS

The distributed command & control environment is especially complex and may certainly exhibit dynamically changing attributes during its operation. Many critical applications within this environment require specified levels of service from the distributed resources. Therefore the direct implementation of these quality of service features into the distributed command & control infrastructure can be extremely advantageous for software programs which have these mission critical requirements. To properly ascertain that the essential quality of service based system resources are being reasonably utilized and efficiently shared among these programs some evaluation of the resource distribution method should be conducted. However, current analysis techniques for evaluation of quality of service specifications are somewhat lacking in that there is no exacting method to determine precisely what quality of service conflicts take place during actual program execution.

This problem within our focus area of distributed command & control environments requires a proper working definition of command & control. However, defining a command & control system is not as straightforward as one would anticipate. There is considerable debate within current literature on the exact meaning of command and control, and C3, C4 systems [Boyes 97]. Certainly, there are entire disciplines dedicated to the study of command & control and much time and effort has been devoted to this work. The intent of proposing a definition of command & control is not to detract from this valid effort, but instead to narrow the scope of the interpretation to a workable proportion. This will allow for some congruity of the terms within the context of this research.

An integral definition is proposed by [DOD 00] which states that command and control systems include the facilities, equipment, communications, procedures, and personnel essential to a commander for planning, directing, and controlling operations of assigned forces pursuant to the missions assigned. Command, Control, Communications, and Computer Systems are defined in [DOD 00] as the integrated systems of doctrine, procedures, organizational structures, personnel, equipment, facilities, and communications designed to support a commander's exercise of command and control across the range of military operations.

Unless otherwise noted, for the purposes of this research work the more comprehensive definition found in [DOD 00] will be utilized. The term command & control systems shall be synonymous with command, control, communications, and computers systems even though the latter term certainly connotes a significantly broader scope by including the integrated systems of doctrine, and organizational structures. Nonetheless, these systems are all considered useful in allowing the commander to broaden his region of awareness in a typical theater of conflict. We must also keep in mind that command & control systems exist to serve the commander as a means to make insightful decisions and observe the resulting consequences of these decisions.

Various command levels exist within typical generalized command & control environments [Boyes 97], as shown in Figure 1[Cooper 94]. Each of these levels involves varying degrees of command involvement. For example to command weapon direction, state of the art automation has enabled much simplified transactions and a substantially lower measure of required input for correct functioning to occur. The amount of commander involvement and feedback increase as the complexity of the action increases, with a significantly greater command involvement seen for strategic planning.

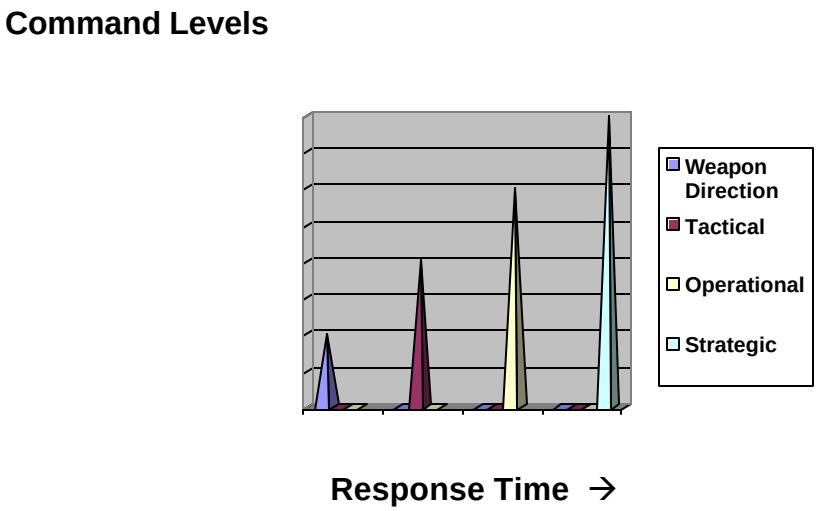

Figure 1. Levels of Command

The regions shown in figure 1 indicating Response Time also suggest an analogous increase as the Command Level increases. This is likewise inline with the increased complexity of the desired action to be taken. This increased complexity can put a strain on the availability of resources and established timing requirements. For example, as command level increases CPU utilization, database access, and effective communication bandwidth resources must be shared among an expanded aggregate of demand that includes increases in both human and technological elements. Likewise, increased transaction processing execution within a specified resource allocation can (and

usually does) require addition completion time for transactions, which can have an adverse effect upon pre-established resource specification requirements.

Furthermore, adding distributed processing activities to the command & control domain also acts to substantially expand the complexity of the environment.

Distributed system environments by nature present an increased impediment to proper resource sharing when compared to non-distributed system environments. As noted in [Barta 95] significant elements which assume magnified concern in distributed systems include: Naming, Transmission, and Scheduling. Naming involves name-to-address mapping usually performed by a dedicated name server. Transmission refers to the transfer of packeted data over some network media through a linearized representation of that data. Scheduling involves a global timetable for observation and regulation of node performance, deadlock and time-out detection. Certainly these elements by themselves are large enough to have dedicated disciplines devoted to their individual study and analysis. All of these elements can coalesce to complicate the accurate quality of service behavior prediction of distributed system environments, which are not typical of non-distributed systems.

Despite this expanded complexity, the augmentation to command & control environments of distributed processing is nevertheless desired due to the potential for improvements in program accessibility and overall performance, additional resources to be shared, and the increased fault tolerance capabilities.

What exactly is distributed processing? The definition of the term “distributed processing” needs to be established and it also is not without controversy. [Kopetz 97] says that a distributed system consists of a set of nodes and a communication network that interconnects these nodes. Others refer to it as any of a variety of computer systems that use more than one computer, or processor, to run an application [Web 00]. We use a

simpler interpretation. For the purposes of this research, any environment that performs computing functions is defined to be distributed if the application software structure and data computations within these environments are dispersed over two or more computing entities. These distributed processing computers communicate by means of some form of interconnected network. Within the context of this research in the area distributed processing the focus is upon the analysis of proper deployment and dispersion of available resources. This direct analysis is carried out though the use of quality of service based behavioral models. The development characteristics of the behavior model are described in chapter II section B.

Needless to say the military distributed command and control environment is a demanding framework that requires transformation as rapidly as the pace with which our contemporary technology advances. The requirements of command and control systems are extremely unstable and changeable due to tactical surprises, changes in goals and missions, etc. [Harn 99]. Currently, an ever increasing level of distributed programs require conventional end-to-end service guarantees. Next-generation distributed real-time applications, such as teleconferencing, avionics mission computing, and process control, require end-to-end systems that can provide statistical and deterministic quality of service guarantees for latency, bandwidth, and reliability [Flores 99]. Programs, such as distributed command & control, which require consistent situation perception for collaborative planning and decision making fit into this taxonomy. These guarantees include elements of bandwidth level requirements, selectable levels of reliability, adjustable jitter, and controlled variable latency levels. The guarantee of a given data flow and the end-to-end service provisions mentioned above can be affected by the proper resource deployment along this flow path. This research investigates these service provisions in relation to appropriate resource utilization as discussed further in this document.

Given all this resource contention complexity and strict requirements the distinct possibility exists for a given situation to be perceived through the utilization of late or

out-of-order data. The resulting false situation assessment could then lead to a flawed or faulty decision being pursued. Therefore distributed end-to-end data flow paths within a command & control system must support some form of efficient resource deployment within specific parameters to achieve a correct situation assessment and avoid a tainted decision making action. Certainly, potential resource utilization difficulties can be attributed to inefficient resource deployment and conflicting program interaction along situation assessment pathways. These undesired and interim complications can lead directly to the late or out-of-order information being pursued.

Essentially this notion of efficient resource utilization can be considered an imperative principle in the coherence of data within a given information flow. This information flow in turn has a direct bearing on the consistency of a given situation perception. A critical ingredient in maintaining an effective flow of cognitive data across pathways within a given distributed command and control environment is the expeditious management of available resources. Specific management and proper interaction of resource elements along these flow paths is of strategic importance in assuring this effectiveness. The basis of these critical interactions can take on many forms and include requirement conflicts and/or resource contention, which must be mitigated through proper resource deployment. Specific applications that demand precise execution considerations can be directly influenced by the explicit level of quality of service that has been assigned by a resource manager for their proper execution.

This research therefore, has the following specific goals:

- Development of practical event models based upon quality of service actions.
- Development of representative high-level quality of service behavioral models focused upon domain specific areas within a distributed environment, which benefit from quality of service treatment.
- Creation of a framework for applying the quality of service behavioral analysis to applications within the distributed command & control environment.

- Application of this framework to perform an effective quality of service event trace upon a targeted application within the command & control domain to isolate potential failure points.
- Development of an appraisal on the influence that quality of service level requirements specification can exert upon software engineering within distributed environments. There will be no specific metrics involved in this simple e.g. good/bad appraisal that identifies specific quality of service actions through observed behavior and describes the potential influence.

As this pursuit of attaining a distinct and efficient quality of service is a major part of this research work, an appropriate formal definition both qualitative and quantitative of quality of service is also in order. The phrase “quality of service” can be, and has been, defined in numerous ways. Various qualitative definitions have been derived from diverse sources as follows.

The Imperial College Department of Computing, United Kingdom [Foldoc 01] defines quality of service as communications and networking which focuses upon the performance properties of a network service, possibly including throughput, transit delay, priority. Some protocols allow packets or streams to include QoS requirements.

In [MIRnet 99] an international consortium of universities and institutes in various countries around the world provides an initial interpretation and discusses a collection of quality of service definitions submitted by member scientists (listed as contributors):

- The following resources deal in quality of service (QoS) as related to advanced networking. The Internet2 definition of QoS is "the ability of an application to receive a predetermined level of end-to-end performance from a network. This may include a particular amount of bandwidth or guarantees of maximum latency or jitter."

- Quality of service (QoS) is a "set of qualities related to the collective behavior of one or more objects." Source: ISO 95 QoS Framework, ISO/IEC/JTC1/SC21/WG1 N9680. Contributed by: Jörg Lieberherr
- Quality of service (QoS) is a concept by which applications may indicate and even negotiate their specific requirements to the network. This can be used for connection admission control, where the network accept or deny new connections based on the given QoS requirements. QoS parameters may also be used for optimizing internal resources, such as choosing less loaded routes in a network. Contributed by: Mike Nahas
- QoS - a multidimensional objective function [defined across multiple service classes] including such parameters as delay (mean and/or specified quantity), call blocking probability, end-to-end time jitter, throughput, and the like. Contributed by: Steve Patek.
- End-to-End QoS services is the ability of a network to deliver service needed by a specific network applications from end-to end with some level of control over delay, loss, jitter, and/or bandwidth can be categorized into three levels of service: Best Effort Service (basic connectivity with no guarantees. The Internet is an example of best effort level of service); Differentiated Service (expedited handling for specific classes of traffic. [...]); Guaranteed Service (a reservation of network resources to ensure that specific traffic gets a specific level of service it requires). Contributed by: Greg Youngblood.

Cisco Systems* talks about quality of service and architectures in [Cisco 00]. QoS can be thought of as measures of performance for a transmission system that reflects its transmission quality and service availability. What is quality of service? QoS refers to the ability of a network to provide better service to selected network traffic over various

underlying technologies including Frame Relay, Asynchronous Transfer Mode (ATM), Ethernet and 802.1 networks, SONET, and IP-routed networks. In particular, QoS features provide better and more predictable network service by: Supporting dedicated bandwidth; Improving loss characteristics; Avoiding and managing network congestion; Shaping network traffic; Setting traffic priorities across the network.

The quality of service definitions described above come from well-respected sources however they share the commonality of chiefly focusing upon the network media as a singular quality of service resource that is typically not coordinated from a singular resource manager. Nevertheless, there are numerous other resources, which also play important roles in the allocation of quality of service levels within distributed systems. Certainly the network media resource can be considered as a critical element of this system. However, to expand our view of true end-to-end quality of service in a distributed environment we must include other resources such as central processing unit, system memory, secondary data storage, etc. which all are positioned within the end-to-end path.

Therefore for the purpose of the research being presented in this document, the term quality of service is to be defined qualitatively as follows: Specific measures of provisioning shared resources in a logical way that can be based upon priorities/needs/requirements of applications. As mentioned earlier this provisioning has typically been foremost implemented within the networking and communication communities. These resource provisioning implementations have commonly followed a decentralized multiple resource management technique to disperse the required communication bandwidth resources at numerous locations (e.g. routers, ATM switches, etc). However, the scope of the research being stated within this document would expand this resource management to include system resources not limited to simple communication elements. Other resources included within this expanded scope will consist of CPU, Network, Disk, I/O, Memory, etc. These resources can also be considered critical elements within a true end-to-end distributed environment and have a

direct bearing upon any quality of service capabilities. Also for the purpose of this dissertation research study the provisioning approach for controlling and coordinating these critical quality of service resources will be centralized within a singular system as can be found within the Linux/RK resource kernel[Rajkumar 98]. This dissertation research does not study the multiple controller communication/network approach mentioned earlier. A more detailed discussion on the Linux/RK system can be found within chapter V section B.

To quantify this quality of service term in a tangible metric form, a set of measurable quality of service parameters have been developed. These parameters focus upon quantitative measurements which include: application quality of service violations(count number of violations per session), quality of service specific message passing (quantity of messages sent /received), quality of service negotiation and re-negotiation requests (number of attempts/failures), quality of service resources and availability(quantity of resources obtainable), quality of service resource requests (number of requests successful/unsuccessful), quality of service resource management actions (allocations/reallocations performed), quality of service level (quality of service desired/achieved). When grouped together these parameters allow a significant analysis of quality of service behaviors. These parameters can be considered somewhat independent, however when assembled collectively an overall quality of service figure of merit can be extracted and may be utilized.

The relevancy of these measurable parameters applies directly to various aspects of the fulfillment of proper quality of service levels. The metric that focuses upon the collection of quality of service violations, can measure the impact on the desired goal of achieving efficient resource distribution. The metric that measures the amount of quality of service specific message passing successes has a direct bearing upon the application attempts to achieve the requested quality of service level with minimal overhead of stale communication. The metric which measures resource negotiation and re-negotiation attempts can be used to determine the efficiency of the resource manager to allocate

resources and can be directly relevant to the attainment of the required quality of service level. The metric that measures the quantity of the quality of service resources with which a resource manager can allocate to the requesting applications it utilized as a basis for calculating resource distribution efficiency and has a direct bearing upon how resources are to be shared. The quality of service resource requests metrics identify the application participation with the resource manager and establish a inception point or basis for measurement of successful quality of service level achievement measurements. The actions of the resource controller in treatment of quality of service assets can be counted for the purpose of determining efficiency in resource allocation and acquisition. Finally a metric that measures the specific quality of service achieved during a typical session is valuable for determining overall success and effectiveness of the system.

These parameters are appropriate and necessary components for measurement of overall quality of service that in turn play a critical role in efficient resource utilization and proper resource management. Here the term proper resource management denotes maintaining resource provisioning amongst requesting applications in a logical way based upon some value function (e.g. priorities/needs/requirements).

The quality of service based event trace approach allows for a detailed quality of service parameter examination. This event trace is utilized as the tool within the analysis framework for collecting the previously defined quality of service metrics and allows for in-depth analysis based upon the previously developed metrics. The specific events to be isolated within these parameters for this research are based upon actions that may have temporal properties (e.g. Event Start, Event Stop) or simply be atomic in nature (e.g. Initialization). This follows the event trace work of [Auguston 98]: "Every event defines a time interval which has a beginning and end. For atomic events, the beginning and end points of the time interval will be the same."

D. PROBLEM SIGNIFICANCE, POTENTIAL IMPACT

As previously mentioned the notion of efficient resource utilization plays a significant part in the coherence of data on an information flow, and has a direct relation

on the consistency of the situation perception. Providing proper and effective quality of service resource deployment for a distributed command and control environment is crucial to preserving a usable data flow. Effective control and proper interaction of the resource elements within the flow paths can support this efficient resource utilization. Anomalous interactions include requirement conflicts and resource contention that may be alleviated with effective resource deployment and appropriate quality of service levels.

Despite this, the critical element of effective deployment of available resources has been given minimal attention at best during the development of command and control systems. Some of the integral activities performed by resource management systems are illustrated in Figure 2[Lounsbury 99].

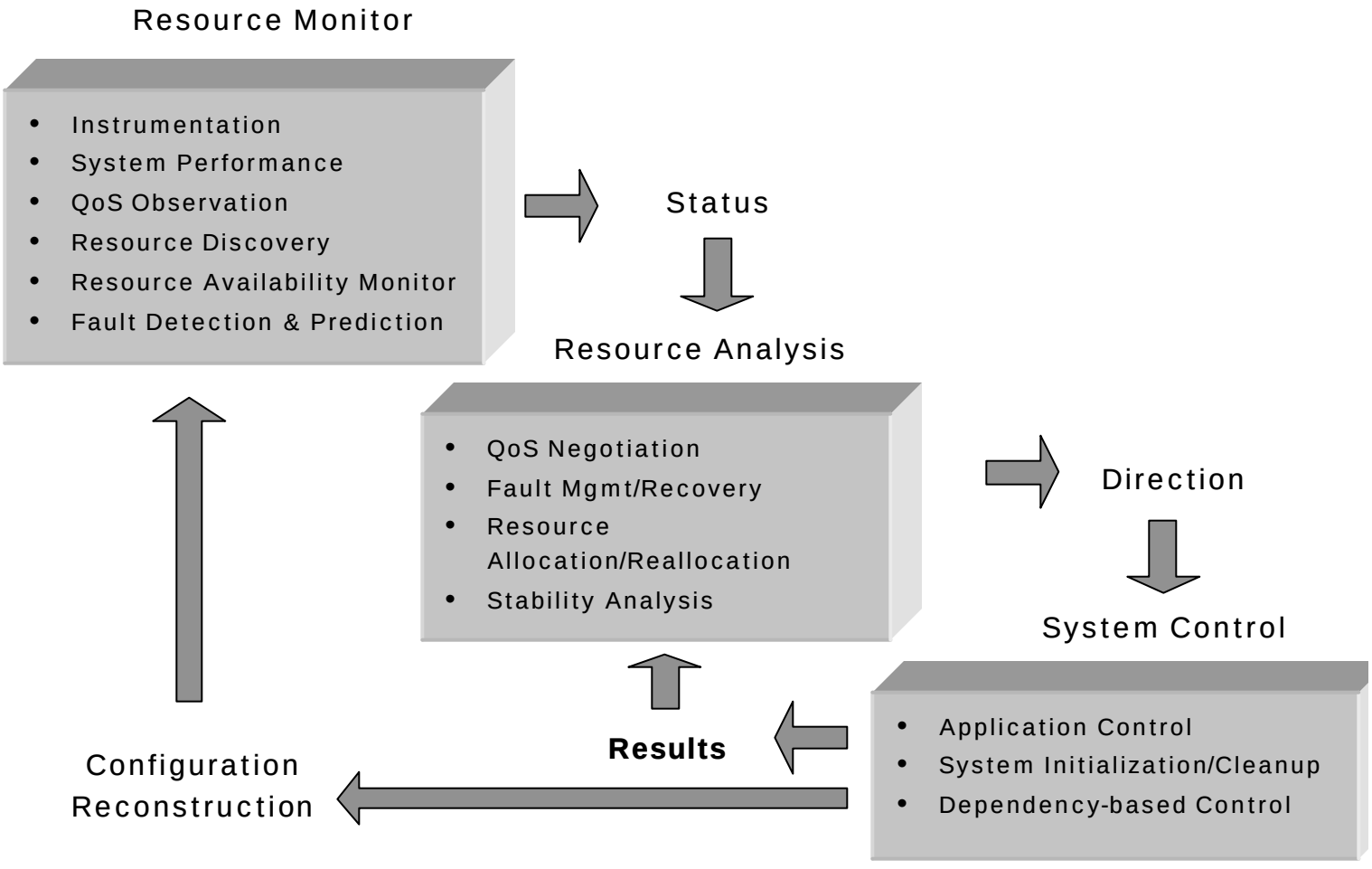

Figure 2. Resource Management

The proper employment of system resources is a major element of achieving appropriate quality of service and effective resource control within a specific information infrastructure. This management of available resources can be approached from a multi-layered standpoint. As illustrated in Figure 3[Lounsbury 99] the scope of resource management in the multi-layered approach covers many domains.

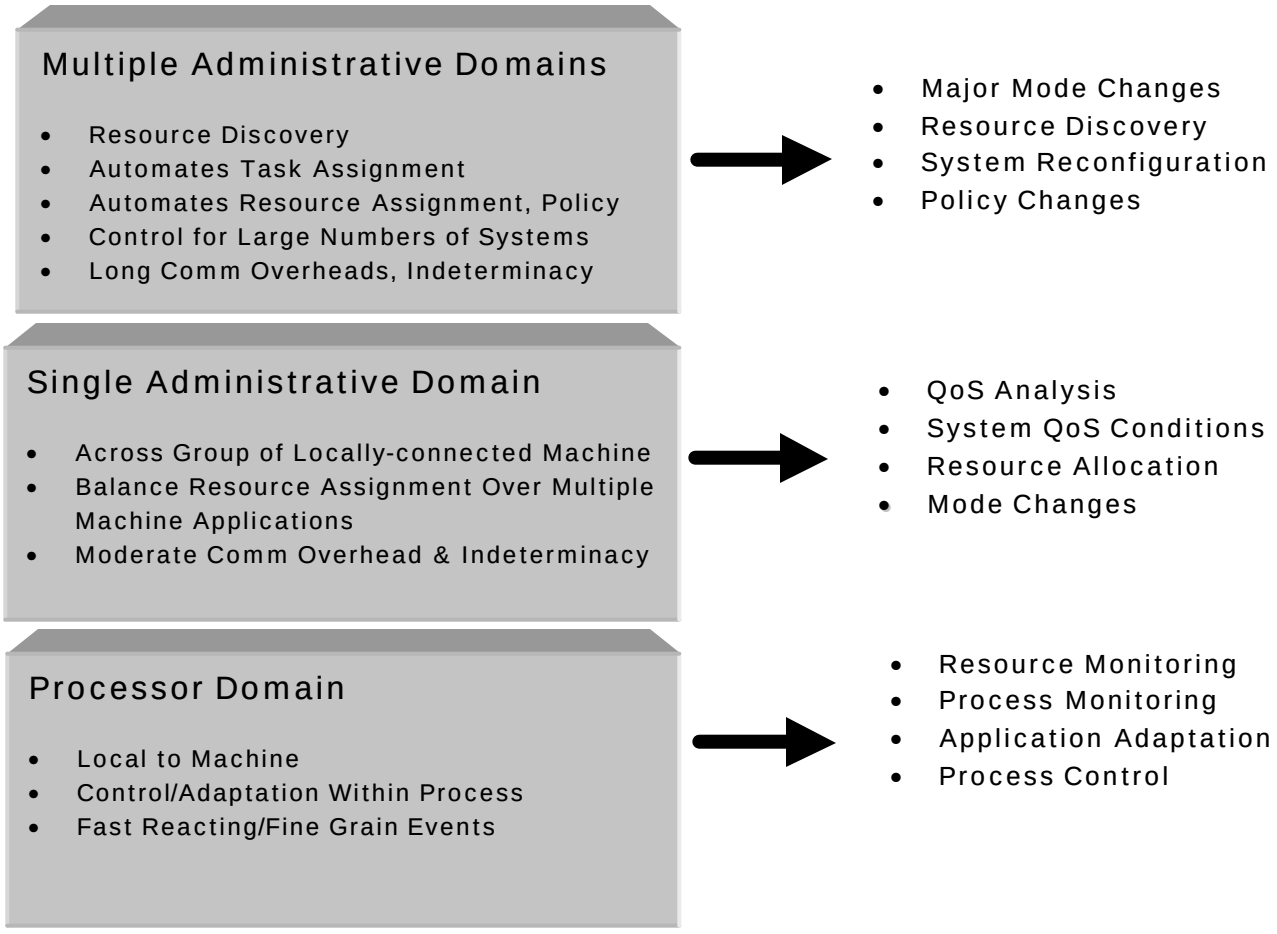

Figure 3. Resource Management Scope

Again, this notion of effective data delivery through proper deployment of resources is a principal element in assuring that stale or out-of-order data does not persist within a given situation assessment pathway. The overall flow of cognitive information must retain its integrity and coherence in order to be useful in collaborative planing and the command decision-making effort. At issue in maintaining the integrity and coherence of cognitive information is the critical ingredient of timely information flow. To assist in achieving this timely information flow it is necessary for the employment of proper resource allocation and the prevention of resource contention within the system domain. This will act to alleviate the difficulties of decayed or out-of-order data due to resource deployment inconsistencies within this information flow. To determine if these resources are being properly allocated a quality of service analysis of the system is desirable.

THIS PAGE INTENTIONALLY LEFT BLANK

II. OVERVIEW

A synopsis of the general development of quality of service behavioral models for distributed system programs and the basic analysis approach to be followed is presented in this chapter. Additionally the detailed method for producing the quality of service analysis framework for application to the targeted program is discussed. The specific focus areas are outlined in the sections below.

A. RESEARCH STRATEGY AND APPROACH

The approach in this work is based upon the presumption that systems within a distributed end-to-end command and control path should support some form of negotiated quality of service levels through proper deployment of all available resources to ensure proper execution of mission critical distributed command & control programs. This focus of accurate quality of service is a much needed element for current DOD systems as stated by the Defense Advanced Research Projects Agency Quorum program manager [Koob 99] “While emerging network-level QoS mechanisms (such as RSVP) are an essential enabling technology for Quorum, they are insufficient in that they are limited to communications QoS. Quorum defines "end-to-end" as being the quality-of-service seen by the application, which calls for coordinated QoS management across middleware, operating systems, and networks.” Additionally, as mentioned earlier, without the proper timely management of these resources there exists a distinct possibility for a command and control situation to be ill perceived because of the utilization of late or out-of-order input data. Compounding the problem, this false situation assessment may lead to a flawed decision being pursued.

B. TACTICS FOR PRODUCING NEW CONTRIBUTION

This research work begins with an investigation of how a specific system can aid in the mitigation of resource utilization difficulties within end-to-end quality of service environments. This investigation is accomplished by an in-depth look at various quality of service and resource deployment characterizations as well as the application of high level modeling and a quality of service analysis framework. The detailed analysis is attained through the utilization of the SPAWAR System Center DARPA Quorum Integration Test & Exploitation project (Quite) testbed environment located at the SPAWAR System Center. Additional details regarding this testbed can be found in chapter V. Testbed Environment.

The specific emphasis of the DARPA-ITO sponsored Quorum program is the development of computing environments with quality of service attributes, controls, and guarantees on local to global scales[Koob 99]. This Quorum program is a multi-million dollar collaboration of fifty top research groups representing universities, industries, and SPAWAR System Center. The research work reported here leverages from this DARPA project focus and this association has provided a high degree of valuable input.

The area of concentration within this targeted environment is based upon the following problem domain characteristics: Distributed computing environment, multiple heterogeneous systems, network medium connections, software applications with specific requirements (quality of service resource needs), centralized resource control software (with quality of service awareness), metrics data gathering instrumentation software. This specifically isolated computing environment can be readily encountered within the AEGIS system, as well as in many other DOD and commercial systems. AEGIS is a combat system architecture that contains a computer-based command & decision component. The core element of the AEGIS architecture provides simultaneous operations capability. These operations include measures against multi-mission threats

including anti-air, anti-surface, and anti-submarine. The problem space of this domain requires specific levels of performance to operate correctly.

The generalized tactics for this dissertation research work includes the creation of a framework for applying the quality of service behavioral analysis to applications within the distributed command & control environment as illustrated below.

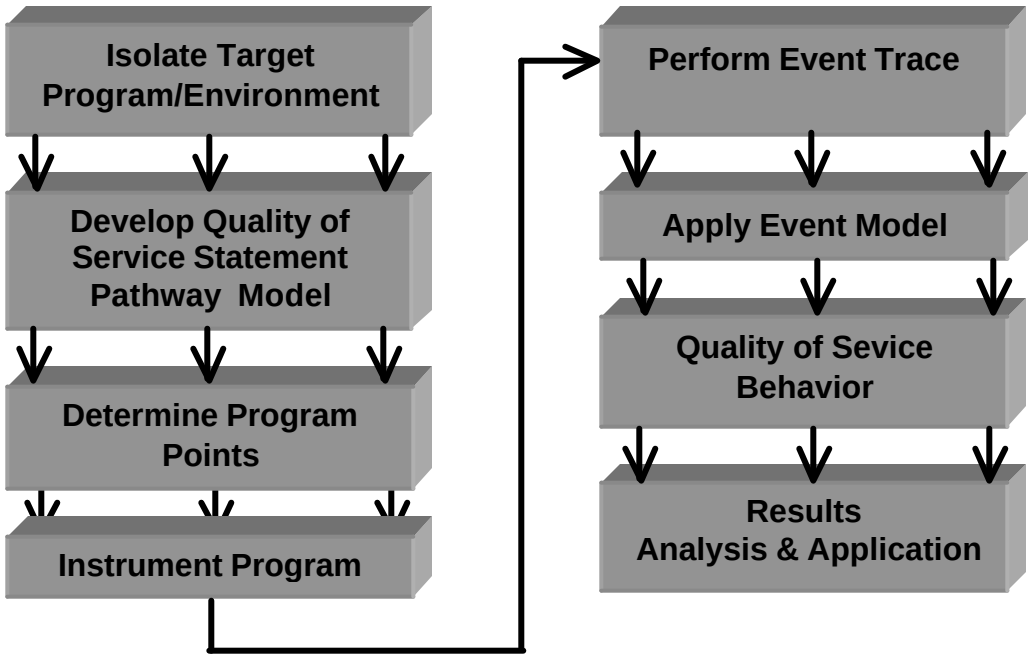

Figure 4. Quality Of Service Analysis Framework.

The creation of the quality of service analysis framework adopts the following approach sequence:

- Select a typical distributed command & control environment.
- Isolate a specific element of a typical command & control environment, such as tactical or strategic, and select a target program.
- Develop operative models of execution pathways (quality of service & resource management specific statement execution) within the target

application in this specific element based upon identifiable details such as resources requested, resources utilized, resources available, etc.

- Initiate the development of a working model of program behavior based upon quality of service factors. This is accomplished by producing abstractions of events that are fundamental to specific quality of service actions performed during typical program execution, which include:
    - Quality of service request statement execution that requests resource reservation.
    - Procedure execution that focuses upon the evaluation and negotiation of available resources to be applied to the originating resource request.
    - Software statement execution of procedures for proper utilization of the assigned resources.
    - Execution of statements responsible for the detection of any resource needs change within the application software.
    - Execution of procedures focusing upon the re-negotiation based on increase or decrease of available and previously assigned resources.
    - Execution of reallocation statements for specific resources by the resource controller.
    - Sending and receiving of quality of service related messages by both the application and resource controller software.
- Identify quality of service specific application program points that directly relate to appropriate resource deployment as illustrated in Figure 5. Such elements have direct consequences upon quality of service behavior and include:
    - QoS specific message passing

- Application QoS violations
- QoS negotiations
- QoS resources and control of resources
- QoS re-negotiations
- QoS level

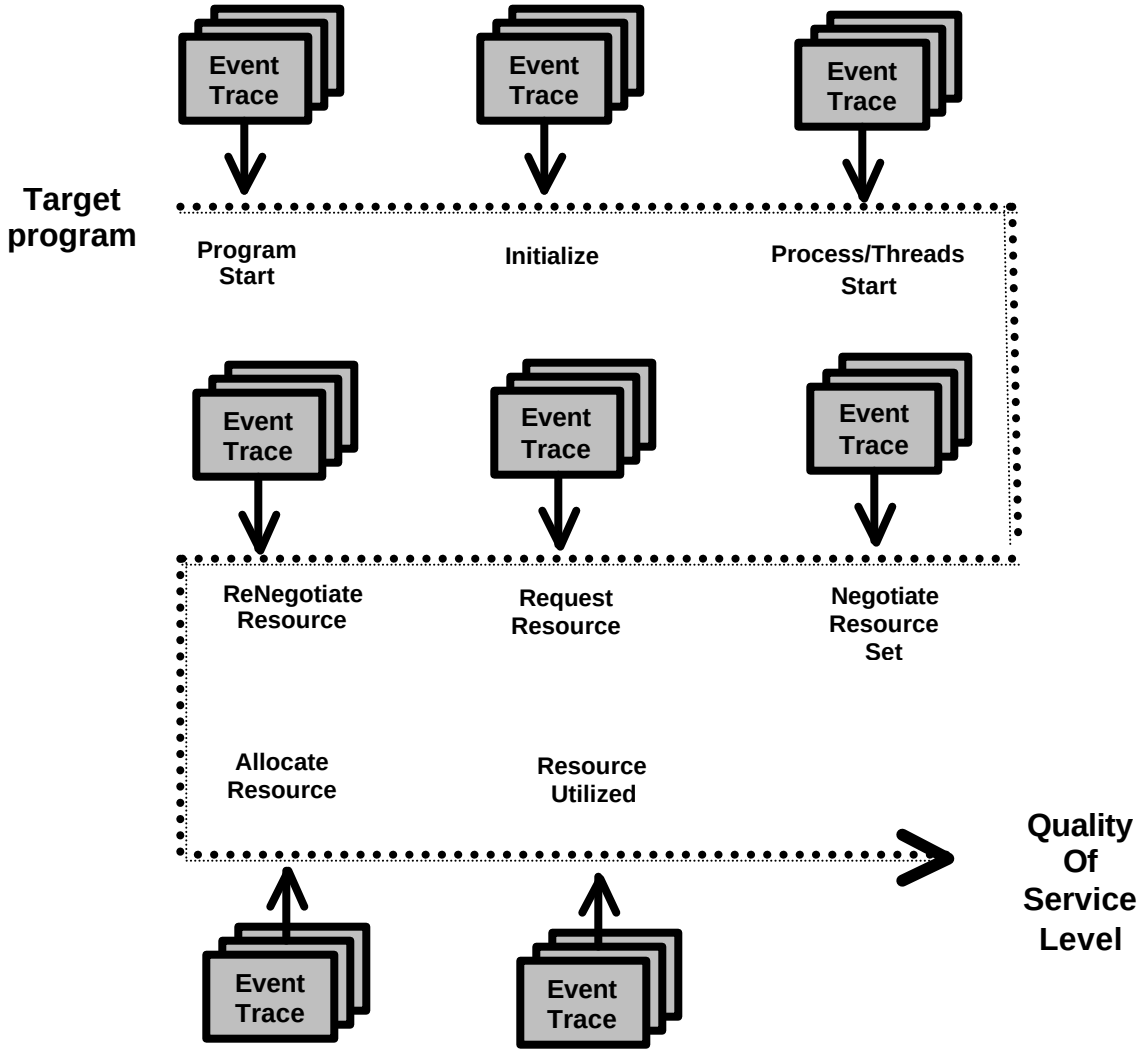

Figure 5. Quality of Service Program Points.

- Instrument the targeted program based upon these previously identified specific quality of service program points. This direct code instrumentation will allow for effective event trace recording at the precise location of the quality of service actions of interest.
- Perform quality of service event trace analysis upon the executing targeted program using this instrumentation.

At this point it is necessary to further expand upon the events of interest for quality of service analysis and the development of the behavior model. The smallest element of the behavior model is the event. An event is a detectable action that influences the overall achievement of the desired quality of service level. The discovery of this action is noted by the embedded instrumentation within the targeted sources as previously shown in Figure 5. The event attributes include the process or thread within which this event has occurred (in the specific case of the targeted environment both thread and process are noted), and a boolean attribute denoting the associated quality of service action of success/failure.

The event model is constructed from a specific quality of service based action and the attributes relevant to this action. The event model is applied to the event trace for the purpose of constructing the quality of service behavior. There are eight events of interest and their respective attributes that form the composition of the behavioral model.

The resource request event is critical to the development of the behavioral model because it represents the applications mechanism for acquiring the proper resources for successful program execution. Within the quality of service behavioral model every resource request event and subsequent failure/success attribute is indicative of the applications behavior. The resource request event model is composed of the action of requesting resources and the set of event attributes that include: depth level ‘DP, process

type ‘TP, location ‘LC, path ‘PA, resource type ‘RT. The request resource event RQ = {DP, TP, LC, PA, RT}, this event model is illustrated in the next figure.

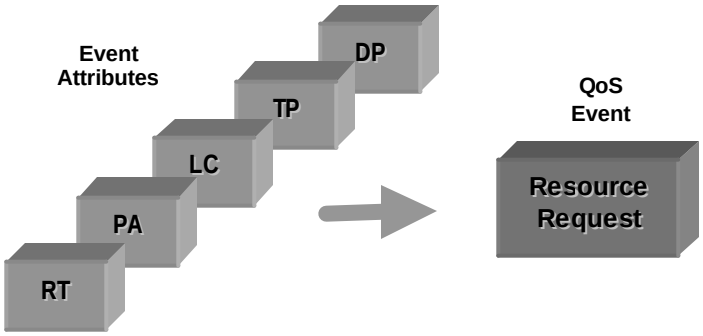

Figure 6. Resource Request Event Model.

The quality of service violation event is an important element of the behavioral model as it is representative of a quality of service fault. This failure event has a causal relation to the preceding quality of service associated attempt actions that include resource negotiation, resource request, resource re-negotiation, and resource assign. Within the composition of the quality of service behavioral model failure is indicative of the applications behavior. The quality of service violation event model consists of the resource request failure, or resource negotiation failure, or resource re-negotiation failure, or resource assigned failure actions and the set of event attributes: depth level ‘DP, process type ‘TP, location ‘LC, path ‘PA, resource type ‘RT. The quality of service violation event QV = {RT, DP, TP, LC, PA} and this event model is illustrated here.

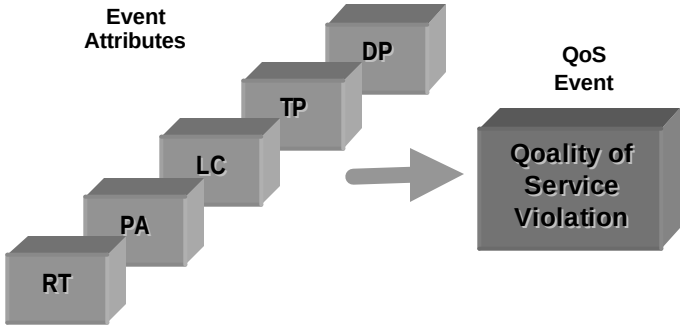

Figure 7. Quality Of Service Violation Event Model.

The quality of service level event supports the behavior model as it represents the action of the resource controller appropriating the requested resource. Within the quality of service behavioral model the appropriation of resources is a significant action in the attainment of proper quality of service and consequently characterizing the applications behavior. The quality of service level event model is composed of the action of resource reserve creation success action through the resource controller and the set of event attributes that include: depth level ‘DP, process type ‘TP, location ‘LC, path ‘PA, resource type ‘RT, resource size ‘SZ, resource period ‘RP, resource deadline ‘RD, and resource used ‘RU. The quality of service event QL= {DP, TP, LC, PA, RT, SZ, RP, RD, RU}, this event model is illustrated in the next figure.

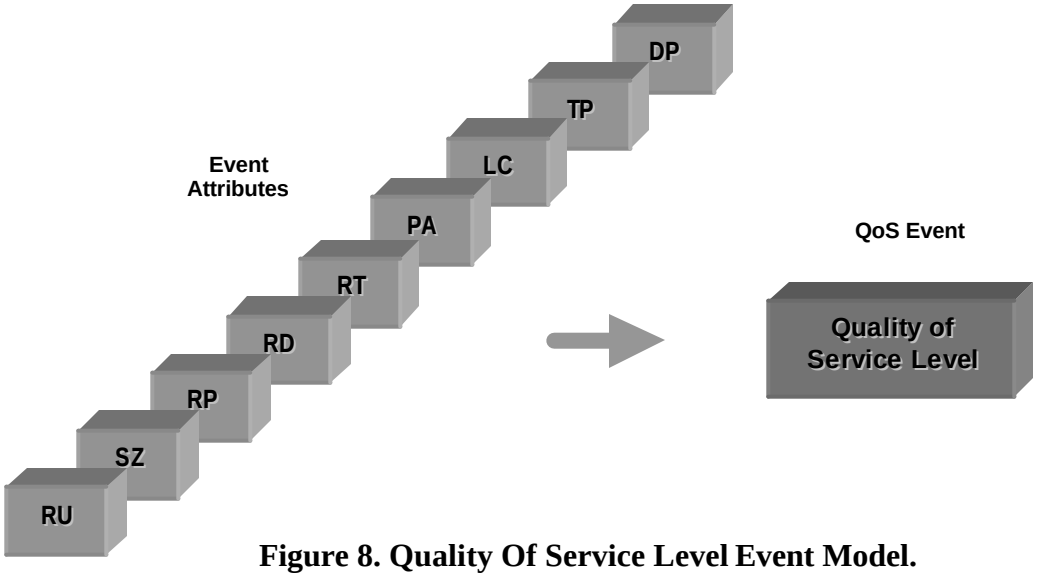

Figure 8. Quality Of Service Level Event Model.

The resource negotiation event is an important part of the behavior model because it represents the transaction of establishing a resource set with the resource controller. This event is significant in the acquisition of resources and therefore an important in the development of the applications quality of service behavior. The resource negotiation event model is comprised of the action of setting up this resource set and the event attributes that include: depth level ‘DP, process type ‘TP, location ‘LC, path ‘PA, resource type ‘RT. The resource negotiation event RN= {DP, TP, LC, PA, RT}, this event model is illustrated in the next figure.

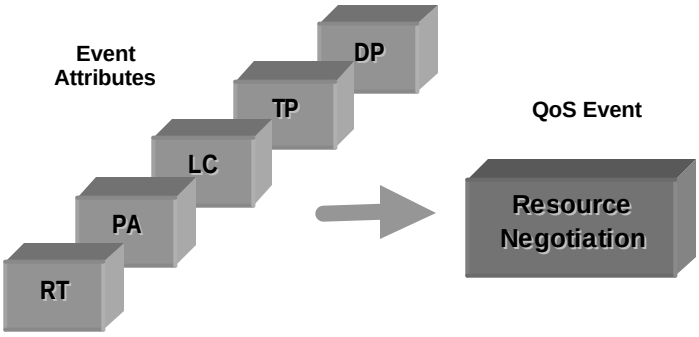

Figure 9. Resource Negotiation Event Model.

The system reservation event is a component in the behavior model because it represents the action by the system, and other applications not currently being targeted by the event trace, of requesting resources from the resource controller. When the focus is directed only at the target program for evaluation the system resource event simply represents a competing load application. This event is critical to the quality of service behavior model because it enables an evaluation of the target program under resource competition load. The system reservation event model consist of this resource reservation action by these competing users through the resource controller and the set of event attributes that include: process type ‘TP, location ‘LC, path ‘PA, resource type ‘RT, resource size ‘SZ, resource period ‘RP, resource deadline ‘RD, and resource used ‘RU. The system reservation event SR = {TP, LC, PA, RT, SZ, RP, RD, RU}, this event model is illustrated below.

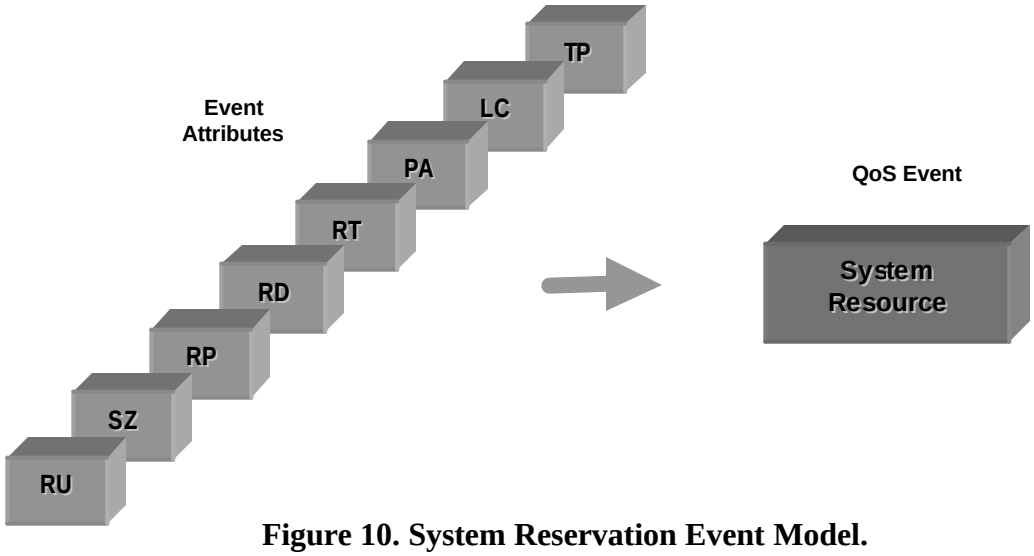

Figure 10. System Reservation Event Model.

The resource assignment event is a critical element in the quality of service behavior model because it describes the action of assigning resources by the resource controller to the requesting thread/process. The resource assignment event model is composed of the action of attaching the resource set to the specific process/thread through the resource controller and the set of event attributes that include: depth level ‘DP, process type ‘TP, location ‘LC, path ‘PA, resource type ‘RT. The resource assignment event SR= {DP, TP, LC, PA, RT}, this event model is illustrated in the figure below.

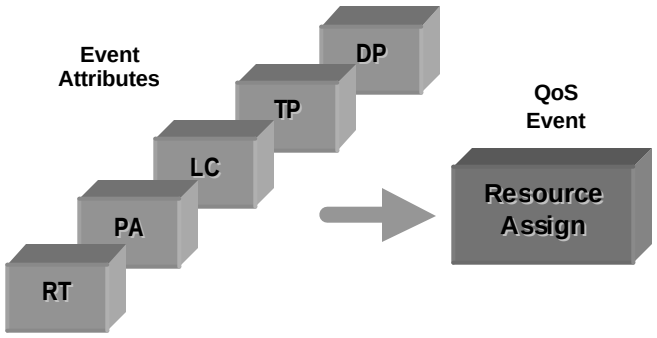

Figure 11. Resource Assign Event Model.

The path length event is part of the quality of service behavior model because it represents the action of traversing the quality of service path to a succeeding program point level within the event trace. The path length event model consists of the action of proceeding through program points and the set of event attributes that include: depth level ‘DP, process type ‘TP, location ‘LC, path ‘PA. The path length event PL = {DP, TP, LC, PA} and this event model is illustrated below.

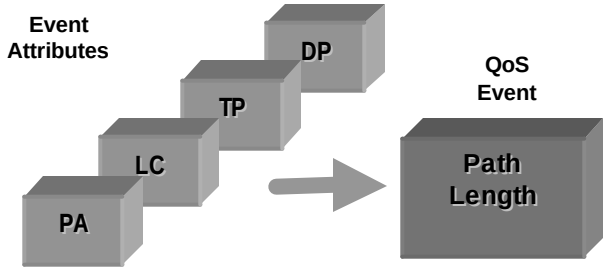

Figure 12. Path Length Event Model.

The resource re-negotiation event is critical to the quality of service behavior model because it is representative of the process/thread actions to correct a preceding quality of service violation. The resource re-negotiation event model is comprised of the action of re-negotiating the amount of resource requested through the resource controller and the set of event attributes that include: depth level ‘DP, process type ‘TP, location ‘LC, path ‘PA, resource type ‘RT, resource size ‘SZ, resource period ‘RP, and resource deadline ‘RD. The resource re-negotiation event RR = {DP, TP, LC, PA, RT, SZ, RP, RD}, this event model is illustrated in the figure below.

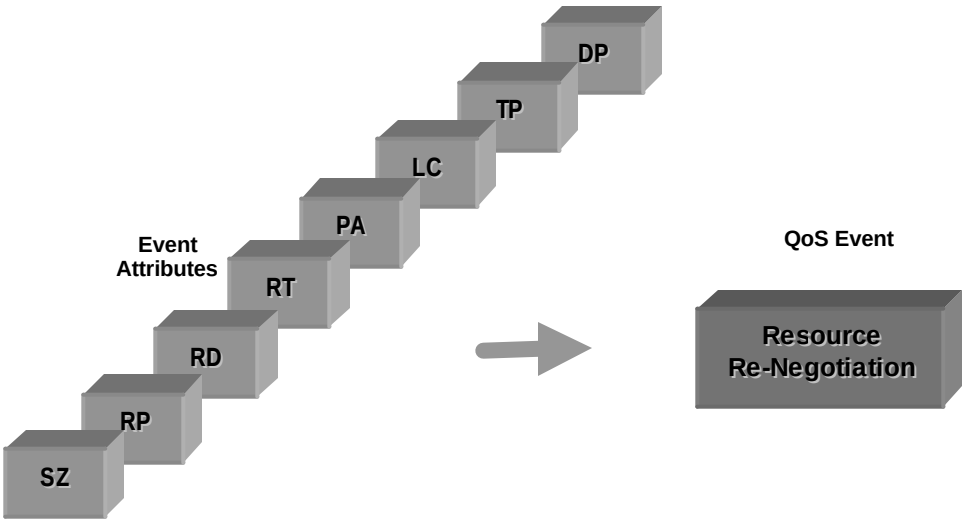

Figure 13. Resource Re-Negotiation Event Model.

These specific event trace quality of service actions which comprise the event models are informally structured with no imposed overall ordering structure as they occur asynchronously. However, there is a partial ordering within a specific thread/process execution as denoted by the thread/process depth level attribute, and a causal ordering between request events(resource negotiation, request resource, resource re-negotiation, resource assign) and fault event(quality of service violation). After the event occurs the event trace notes the specific attributes of the quality of service action such as

type(quality of service resource), level(path depth), path(aggregate progression of the event trace), ptype(process or thread), and loc(within the process/thread). This data is noted and a boolean evaluation process determines its success/failure attribute.

The behavioral model is composed of resource request event ‘RQ, resource negotiation event ‘RN, resource assign event ‘RA, quality of service event ‘QV, resource re-negotiation event ‘RR, quality of service level event ‘QL, system reservation event ‘SR, and path length event ‘PL. Potential failure behaviors are comprised of the following sets of events: {RN, QV}, {RQ, QV}, {RQ, QV, RR, RR, RR, QV}, {RR, QV}, and {RA, QV}. Typical success behaviors are composed of sets of events that include: {RN}, {RQ, QL, RA}, {RR, QL, RA}, and {RA}. An example of a quality of service behavior model that characterizes quality of service failure is illustrated below.

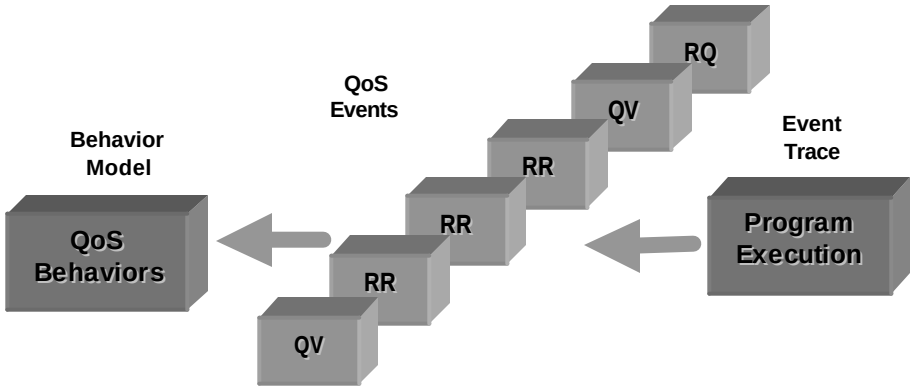

Figure 14. Quality Of Service Behavior Model.

The behavioral model can be utilized to isolate specific quality of service behaviors. For example in the failure occurrence illustrated in above with the events {RQ, QV, RR, RR, RR, QV} the program point of failure can be isolated through examination of the event trace results. This search can be achieved through a retrace of the execution path to the specified depth level of the distinct(named) thread/process, and by examination of the event attributes associated with the failure event quality of service violation QV = {DP, TP, LC, PA, RT}.

The quality of service event trace on the targeted application collects these events and based upon this information the quality of service behavior can be constructed. The behavior model is partially ordered set of event types (e.g. resource request) and event attributes (success/fail boolean). The quality of service metrics of the event trace are calculated based on quality of service actions. These calculations include elements such as the number of resource re-negotiation events that have occurred, and the number of application quality of service violations as discussed in chapter IV. event trace analysis. The results of these event trace metrics include the lists of specific processes/threads identities and their resource specific events. This information provides the necessary data to construct the quality of service behavior.

The next phase in the application of this quality of service event trace framework is to perform this event trace based upon these event models using a target application program that has been instrumented for accurate feedback. This analysis is then utilized to develop an overall quality of service behavior for characterization of command & control systems applications that can be applied to mitigation of discovered quality of service related efficiency problems. Through direct examination of the event trace results the specific potential failure region (QV events) can be isolated to specific path locations and thread/process depth levels that denote the distinct program point within the targeted application. The potential error region can then be adjusted to improve the quality of service level achievement probability.

This discussion has focused upon the general approach to the quality of service event trace. The behavior model of this generalized(general case) framework approach is applied to the specific case implementation as discussed in chapter VI. Event Trace Analysis.

Specific logical conditions and constraints for this work include distributed systems, heterogeneous environment, multiple diverse quality of service levels essential

for program execution (i.e. application requirements vary high/low needs), and available resources can include network bandwidth, CPU, memory, etc.

During this research the development of the specific models is based upon the above properties that could affect bandwidth levels, levels of reliability, jitter, and controlled variable latency levels. As mentioned in [Kopetz 97] "Information reduction, or abstraction, is only possible if the goal of the model building process has been well defined. Otherwise, it is hopeless to distinguish between the relevant information that must be part of the model, and the irrelevant information that can be discarded." As stated in [Welch 99] "it is important to understand the characteristics of the environment of the real-time control system." The effort pursued by Dr. Welch declares that from an engineering point of view it can be beneficial to develop environmental models as ether deterministic, stochastic, or dynamic.

Based upon the previously mentioned investigation goals for this dissertation research effort, these specific quality of service pathways that can exhibit deterministic, stochastic, or dynamic properties are isolated and studied prior to the development of the behavioral models. The abstract events that form the constructs of these pathways (as related to quality of service) are examined in detail to segregate any possibility of resource contention and other adverse conditions. This quality of service path model therefore can be extracted from the overall system architecture.

This description of a pathway is very similar to an abstract scenario. A scenario is described in [Rumbaugh 91] as " …a sequence of events that occurs during one particular execution of a system." Given a specific use case, as defined by [Maher 96] as "Use cases are analogous to the script of a play. They describe the role of each actor in the play and how the actors behave in a scenario." The pathway description discussed here differs from a typical scenario based upon a use case in that the pathway describes a general

instance of an element of a command & control system whereas a scenario would be representative of a narrowly specific focus of the system under scrutiny.

THIS PAGE INTENTIONALLY LEFT BLANK

III. PREVIOUS WORK

This chapter presents a detailed assessment of previous work in the areas of quality of service evaluation and analysis. This evaluation of earlier work also includes explicit appraisal of research in program behavior analysis related to quality of service. This previous research examination also includes a focus upon specific quality of service related analysis of command & control environments and these research efforts are also discussed at length in this chapter.

A. ASSESSMENT OF WORK

The notable work in the areas of quality of service analysis & specification, distributed system modeling, and resource management specification is currently focused upon the development of correct representations of the specific environment or system under investigation. To this end, some researchers take a formal approach, some use informal procedures, some examine the details of truly distributed environments while others focus directly upon the quality of service specific issues that are related directly to the system resources. These representations can, in particular instances, be analyzed for proper resource module interaction and/or appropriate resource utilization.

Regardless of the analysis method that is employed by these capable researchers the typical end goal of their examination efforts are generally comparable. In most cases their common overall objective is that eventually the result of their diverse analysis endeavors may be properly utilized and applied to the specific design of a system for the production and/or development of an accurate and functional architecture, program, or system.

B. RESULTS SUMMARY

In [Montiel 93] methods have been proposed for quality of service verification and testing. Specific network protocols are used for application methodology guidelines and experimentation including XTP and JVTOS. Here the term verification is stated in three areas as: 1. Properties Verification "this is the decision process that establishes the validity of an assertion on the system to which it is applied." 2. Conformance Verification "this is an assessment of compatibility of a system or one of its representations against another representation." 3. Verification in Use "this is the decision process that establishes the validity of an assertion on the status of the system during its use." This work also develops quality of service parameters and metrics necessary for verifying proper quality of service. An extensive breakdown of measurable elements of quality of service is presented. Quality of service categories have been stated which include elements such as Speed (transit time / latency / jitter), Accuracy (message degradation transformation), Robustness (MTBF / graceful degradation / congestion), Protection (information privacy), Capacity (bandwidth / loading / peak-average ratios), Operability (ease of use), Accountability (user specified QoS requirements), and Resource Optimization (dynamic reallocation). This work is very complete. However, it focuses exclusively upon Integrated Broadband Communication (IBC) systems and deals with verification and testing of optimizing techniques of telecommunication media. In contrast to this approach, the quality of service event trace analysis technique goes beyond the focus of a singular resource telecommunication domain. Montiel's quality of service verification methodology, including the developed quality of service parameters & metrics, is very specific to the network resource that he is verifying and is not easily applicable to other system resources such as disk, cpu, and memory. The event trace analysis framework, in contrast, will allow for examining quality of service issues across a multitude of resources. This facilitates the formulation of an accurate and representative behavioral model that is specific to a broad range of quality of service issues.

The work of [Cicalese 99] has suggested an approach for behavioral analysis and specification of distributed software. They have created a program based upon the extension of Java called behavioral specification of distributed software component interfaces (Biscotti). "For each assertion, the Biscotti compiler adds two methods to the bytecode. The assertion evaluation method throws the appropriate exception if the Boolean assertion is false. The assertion chain method iteratively calls other assertion chain methods at the level above it until it reaches the root of the inheritance hierarchy." The augmentation of distinctive bytecode allows the assertions to be made available to the compiler and facilitates the addition of behavioral constructs to Java. This assertion evaluation is ultimately directed toward the achievement of proper software component interaction within distributed systems and as an aid to component interface developers. The instrumentation is also performed within this environment "The Biscotti compiler also instruments the class and interface methods with calls to the appropriate assertion-checking code. Calls to the invariant and pre-condition chain methods, if these methods exist, are added at the beginning of each method. Calls to the postcondition chain method and invariant method, if these methods exist, are added before each method's return point." The behavioral specification followed by this approach utilizes the Design By Contract [Meyer 97] model. "This model views the relationship between a class and its clients as a contract that takes the form of assertions: Boolean invariants, preconditions, and postconditions." This is a good program for behavioral specification within distributed environments though the main focus here is on reusability of software components within these distributed system environments. Additionally this effort is based specifically with the object oriented(Java) environment. However, this generalized contract approach could be utilized in future work to significantly expand the quality of service event trace analysis technique by providing a more formal approach to the development of the quality of service behavioral model.

The work pursued by [Staehli 95] examines quality of service specification for proper resource management. The formal specification implemented for this effort is based upon the utilization of the Z specification language and consists of three parts: Content, View, and Quality descriptors. This is a good approach that initially defines an

IDEAL quality of service specification using this Z language and compares this specification with a logically isolated quality of service availability specification. This approach allows for analyzing the correctness of quality of service management algorithms. However, this work focuses exclusively upon multimedia environments and continuous media, what's more these environments are not necessarily of the distributed variety. Additionally the development of the IDEAL quality of service specification is based upon fixed input-output criteria and susceptible error rates that deal mainly with presentation quality issues. In contrast, the quality of service event trace approach focuses upon exact quality of service actions such as resource reservation, and resource negotiation/re-negotiation. This provides a greater detail of true quality of service efficiency which, can be exercised across multiple environments.

In [Barta 95] the work focuses upon the development of specifications and model for truly distributed systems. This research provides a good discussion on the differences between distributed and non-distributed systems. This approach also accurately defines the specification of system behavior as an isolation of the static and dynamic properties. Barta states "Besides functionality a system description should also include qualitative and quantitative temporal aspects. Qualitative requirements are necessary to express that a particular sequence of actions must be followed. Quantitative constraints define –in terms of units of time how fast the system will react to a specific input. Omitting temporal constraints leaves the specification incomplete …" This work presents a proficient approach to modeling distributed systems. However, this system model analysis emphasizes primarily client-server models. This limited concentration confines the analysis to client-server interactions while avoiding the other distributed environment models sometimes exhibited within command and control environments areas such as peer-to-peer communication. Also, there is no distinct examination of resource specification or analysis provided by this distributed system model. The quality of service event trace approach, in comparison, focuses precisely upon quality of service specifications and the development of accurate and representative quality of service behavioral models. This focus does not place the exceptional emphasis upon the temporal aspects of the behavior but instead stresses the specific actions that are relevant

to quality of service issues such as resource reservation, quality of service violations, and resource negotiation/re-negotiation.

The work of [Rajkumar 00] provides for proficient real-time quality of service monitoring based upon the Linux/RK resource controller. The quality of service analysis approach can simultaneoulsy monitor multiple resource elements such as CPU, network bandwidth, and current effort to include disk bandwidth. This approach also provides a real-time display of the resource utilization and can be connected to billing services. The efficient linux "proc" function is utilized for the quality of service system statistics. However, this quality of service analysis method does not perform a detailed event trace of the targeted application based upon specific quality of service actions. Additionally this quality of service analysis technique has been established to operate specifically within the Linux/RK resource controller environment and there may be difficult porting issues when applying to other resource management approaches. This quality of service analysis technique provides a good dynamic view of the executing system but does not specifically focus upon quality of service violations, and it is not specifically designed to be applied to the software engineering of programs with resource needs. The quality of service event trace analysis framework in contrast is specifically designed to be readily applied to software engineering problems.

The work of [Frolund 98] develops a specific language for quality of service specification called Quality of service Modeling Language (QML). The language leverages heavily from the Universal Modeling Language (UML). This undertaking provides a good argument that critical and important quality of service prerequisites and needs usually are not addressed during the requirements phase of the software development process. Despite this, quality of service requirements can and do significantly affect the design of components in a given system. The approach suggested here is to utilize the Quality of service Modeling Language to allow for a proper capture of distinct quality of service properties and requirements such as reliability, security, and performance. These properties are then included as a specific element of the overall

system design process. This is a good step toward upgrading a system design to include critical elements. However, this approach does not include any specification that would allow for one to perform specific checking of the end program. This work simply specifies assumed and estimated quality of service levels, resource management, and correct resource utilization. Their specification approach is sufficiently precise but the quality of service parameters may be unrealistic for application to military critical systems. As discussed in their example system the resources are expected to be exceedingly highly available and include an infinite MTTF through the utilization of extensive redundancy. In contrast, the quality of service event trace approach determines the efficient resource utilization and quality of service levels through precise instrumentation and analysis of the targeted system. This information can then be forwarded back into the requirements, design, and implementation processes.

[Blair 98] suggests an approach called Aspect-Oriented programming. In this effort there is a good discussion on the formalization of quality of service. This formalization is based upon a process algebra called Language of Temporal Ordering Specifications (LOTOS). As stated in this work, LOTOS is utilized for specification of the functional behavior of a system and a real-time temporal logic is utilized for the specification of the quantitative quality of service constraints. Additionally, probabilistic and stochastic stream behaviors are specified through the utilization of a probabilistic temporal logic. There is a specific distinction discussed between functional, interaction, and non-functional properties. The focus of this work has been directed at determination of distinctions between functional or qualitative behavior and quantitative behaviors that include quality of service constraints with resource management policies. This work provides an excellent approach to performing an examination of the quality of service constraints of a system design. However this approach may become too abstract by not paying enough attention to important actual quality of service performance issues such as the specific resource request, negotiation/re-negotiation actions. The quality of service event trace approach conversely provides a method to develop an in-depth behavioral model based upon the exact quality of service actions of the target system.

The work of [Auguston 00] discusses the idea that testing and debugging are mostly concerned with program run-time behavior, and states that developing a "precise model of program behavior" becomes the first step towards any dynamic analysis automation. This approach focuses upon the analysis of C programming language programs and associated events. Events are key to this research and are utilized as a basis for building program behavior analysis. This work [Auguston 99] states that an event can occur when an action is attained while the program execution process is underway. This work of Auguston is primarily directed toward overall program improvements that do not specifically cover quality of service issues. However, I believe that this "event trace" approach can be aptly applied to quality of service pathways. Specifically this work can be leveraged towards the area of developing quality of service behavioral models for use within distributed command & control systems. Extending this program analysis work into the area of quality of service analysis could be expanded by the implementation of a language such as D-Spec or FORMAN [Auguston 92] that could apply nicely to this analysis. Here an event could be directly related to when a quality of service resource allocation or resource management action is performed. This extension of the quality of service analysis effort is planned for future work and is discussed in chapter VII. Conclusion.

The U.S. Air Force initiated a program for standardization of methods and techniques for modeling and design/analysis of systems which is called Integration Definition (IDEF). Various excellent IDEF elements have been developed through Knowledge Based Systems Inc., some of which have been incorporated into Federal Information Processing Standards Publications. As noted in [IDEF1X 93] the functional model is the focus of IDEF0, an informational model is emphasized in IDEF1, and the dynamic model approach is accentuated in IDEF2. Other areas of IDEF concentration which are considered as previous work includes IDEF3 that focuses upon process description and IDEF4 focuses upon object-oriented design. IDEF3 allows for capturing behavioral conditions of systems including temporal aspects and process interactions

such as precedence and causality. The IDEF4 object-oriented design approach utilizes graphical syntax and diagrams extensively as a tool for consolidation of important design elements. These approaches are quite valid for their specific purposes and can be useful tools for design and analysis of the systems that they are focused upon. However, the IDEF approaches do not specifically address quality of service issues when capturing behavioral conditions, and does not perform formal specification in these areas. In comparison to the IDEF approach, the quality of service event trace procedure focuses exclusively upon quality of service issues. The direct trace into the program points which, form the resource related behaviors of the system under analysis, provide a behavior model that is specific to quality of service actions.

The work developed by [Hrischuk 99] is based upon the utilization of the ANGIOTRACE approach. This approach allows one to gather the necessary information from a design then develop a model through automation. This approach applies to distributed and concurrent software with synchronous communications, and allows the development of a layered queuing network type model. This work also utilizes a technique called Trace-Based Load Characterization (TLC) which provides extensions into the ANGIOTRACE tool allowing both synchronous and asynchronous interactions to be analyzed.

The research described in [Hrischuk 99] is focused primarily upon communications architectures and network infrastructures. The ANGIOTRACE/TLC approach also works well for specifically analyzing software load interactions within architectures when used in the initial design phase of development. However, this approach promptly becomes brittle and cumbersome when applied to existing design instances. The ANGIOTRACE/TLC approach grows unmanageable when exercised on currently implemented command & control architectures. As noted by Hrischuk “The advantages of TLC are most fully realized if it is part of an evolutionary approach to software development, moving from an early executable design or partial implementation to a complete product.” Additionally the major concentration of this approach is to

describe the behavior of task interactions that are not necessarily associated quality of service issues. By contrast, the quality of service event trace approach focuses directly upon quality of service specific actions. Also, the quality of service event trace approach can be applied to communication architectures which implement quality of service features as well as current command & control architectures in the pre-design and post-design phase of development.

The utilization of Petri Nets to design object interactions are described by the work of [Cheung 98]. The motivation for this work is that efficient interaction of models is sometimes not consistent with the object's lifecycle design as shown in Table 1 below. "A lifecycle model specifies the transition of states and the execution of operation throughout an object lifecycle." This approach establishes a nice model interaction design as derived through Petri Net modeling of the interacting object lifecycle. This work emphasizes the development of a scenario by modeling an event sequence and entering this data into a Petri Net environment. This allows for behaviors to be run through a very efficient consistency check procedure when performed inline with the object's lifecycle model.

00 methods	Lifecycle models	Interaction models
Booth' s method 131	State transition diagram (dynamic view)	interaction diagram (dynamic view)
Coad-Yourdon' s OOA[11]	object state diagram (service layer)	message connection (service layer)
Embley' s method 141	state net object behaviour model)	interaction diagram (object interaction model)
OMT [2]	state transition diagram (dynamic model)	event-trace diagram (dynamic model)
OPEN [I 2]	state transition diagram	sequence diagram collaboration diagram
UML [5,6]	statechart diagram	sequence diagram collaboration diagram

Table 1. Lifecycle models and interaction models of six 00 methods

While this approach will work for new development of a distributed command & control architecture the Petri Net practice described in [Cheung 98] is most useful when applied to pre-design phase of development, and does not employ agreeably to existing architectures. Typical legacy command & control architectures exhibit exceptional requirements and constraints. These preconditions soon become uncontrollable when this approach is utilized. Additionally, many of these legacy command & control systems are not based upon an initial object oriented design. This presents collateral difficulties when attempting to apply this work to the numerous legacy systems being utilized today. In comparison to this approach the quality of service event trace procedures allow for direct application to legacy systems.

The work of [Hariri 95] discusses a bi-level hierarchy approach to distributed system availability modeling. The availability models developed through this technique focus upon hardware, software modules, communication links, and collision delays in addition to resource allocation. A graph-based approach is utilized at the system level for analysis of the availability of distributed programs. A second level utilizes a Markovian technique to analyze component availability. The availability approach presented by the first graph based level can be used to identify bottlenecks within critical components, determine the sensitivity of system availability to hardware and software failures, and optimize the process of improving the availability of a distributed computing environment. In addition, since the failure and repair rates can be calculated for each component in the distributed computing environment the reliability, maintainability, and other time-dependent measures can also be computed using the approach. Furthermore, the links and the nodes of process spanning trees can be labeled in a manner such that the effect of some other performance metrics (e.g. delay, throughput) can be modeled. In contrast, the quality of service event trace approach employs a significantly concise focus directed at specific quality of service actions. This tight focus enables the quality of service event trace technique to concentrate directly upon the resource deployment issues as well as specific resource management decisions. This event trace technique provides a more accurate behavioral model that in-turn allows for a proficient quality of service efficiency analysis.

Research conducted by [Harel 97] looks at the feasibility of utilizing statecharts to model elements within a distributed environment. The statecharts, used for behavioral description, are utilized in association with a subset of UML and an automated toolset called Rhapsody that is capable of model execution and code generation. However, although this approach provides a nice automated procedure for analysis and model interaction, it abandons the resource management decision making process completely. Consequently any quality of service issues must be addressed indirectly and haphazardly.

The work of [Pryce 00] aptly models, and efficiently examines, component interaction within a distributed architecture. However, this work considers components to be interchangeable as a service provider or a service consumer. This imposes undue complexity within the area of resource management. There is also a dependency upon specific transport protocols within the distributed system making this approach extremely brittle and difficult to adapt to heterogeneous environments. "In our model, a component is a reusable element of a distributed program that encapsulates state and/or behavior behind a strict interface comprised of the roles that the component may take in various interaction protocols." Additionally this approach does not consider modeling specific pathways that further removes any analysis of quality of service functionality.

Other work in the area of distributed system timeliness is focused primarily upon the implementation of timely quality of service, however proper resource management and overhead cost structure of this service in a timely way can be considered the paramount issue. Technology currently exists for quality of service, and at a reasonable resource cost, however there is a highly variable costing structure with the bulk of the overhead in management of quality of service, not just making it available [Guerin 98].

Contemporary work also concentrates upon management of the specific path itself to achieve quality of service levels that provide a coherence of data. We need a Path

Information Base to provide info on available paths, and to be able to put it in a separate server rather than adding it to the list of things that routers have to do (they're already busy enough) [Xie, 1998]. This will allow for the organization of a hierarchical pathway structure, but we still need proper management of resource utilization at the individual system level to complete the end-to-end cycle in a timely way.

The research of [Matsui 98] creates QoSPath models that denote resource requests via utility functions to a system resource arbitrator. QoSPath models consider variation in QoS parameter formats due to media type differences, distinct resource handling strategies, and particular quality control parameters.

The QoSPath model provides for the specification of quality of service based upon user/application preferences. There is a differentiation between the user specified QoSPath and the system-level QoSPath. The user (or application) specified path represents the primary quality of service needs from the user perspective. The initial user specified QoSPath is translated into its counterpart the system-level QoSPath which in turn provides request information to the QoS arbitrator. The arbitration process is based upon multiple resource requests from differing applications whose utility values and stream priorities are dissimilar, therefore a normalization process is utilized. The arbitrator decision functions are based upon resource policies regarding issues such as admission control and resource distribution.

The approach of isolating a specific path of functionality for the purpose of diminishing global problems into fundamentally solvable elements is not unique. The work of [Mosberger 97] talks about the abstraction, called path, that is complementary to, but equally fundamental as layering in a modular system. Whereas layering is typically used to manage complexity, paths are applied to modular systems to improve their performance and to solve problems that require global context. Within this work it is stated that a path can be described as a vertical section which traverses the representative

system layers and provides insight that is typically not visible though layers alone. As an example a path could be described as portraying the source to destination dataflow when sending a stored file from storage device to network device via an application and the file-transfer protocol (ftp). This work describes these paths to be like small, or localized, vertically integrated systems.

This work is focused on a specific operating system (Scout OS) and is not broad enough in scope to be applied to all resources typically found within distributed command and control environments. However, the notion of path abstraction is well thought out and can be applied to this research. The traversal of system layers through pathways is a good parallel to the command & control environment situation assessment path model. Other areas of focus include the resource management mentioned in [Mosberger 97]. However this resource management is limited in that it is based upon common adjustment of queue size to alleviate potential memory problems and packet flow jitter.

Research from [Sabata 97] depicted in Figure 15 provides a description of elements which compose quality of service. Specifically within the area of performance metrics: Precision is meant to specify volume related quantities, Timeliness represents the timing requirements for performing a given piece of work, Accuracy focuses upon the errors introduced into the data by transients, and Combination includes the metrics of Timeliness and Precision as mentioned above because these two elements can be considered in throughput measurements. The difference between Precision and Accuracy are similar therefore an explanation is needed. “Precision metrics measure the volume of work, and the Accuracy metrics measure the amount of errors in the completion of the work.”

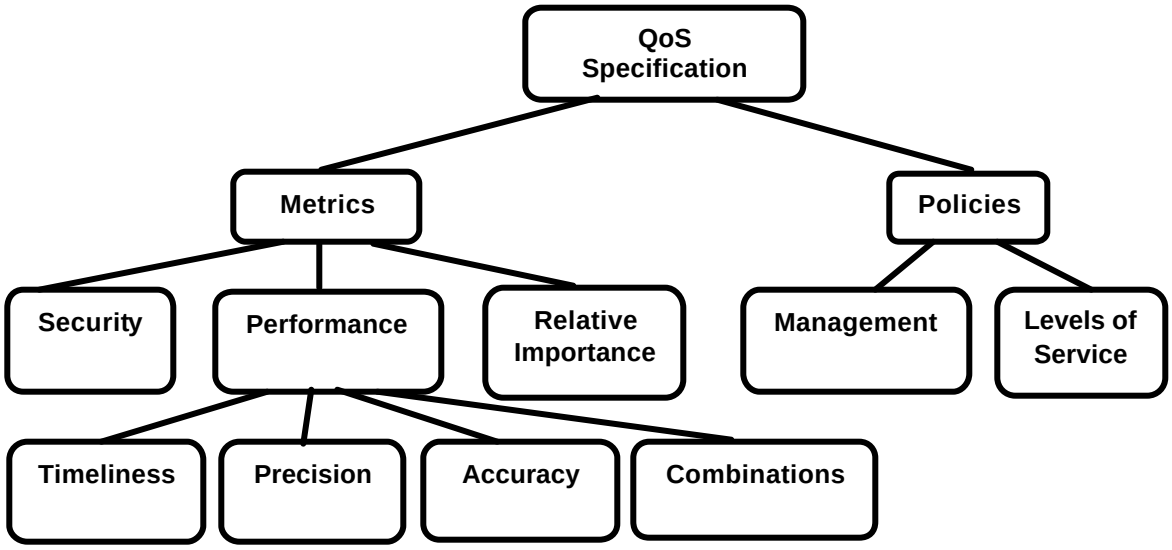

Figure 15. Quality of Service Specification Taxonomy

This taxonomy is a good approach to deciphering quality of service specifications. However, some problems with using this approach should be addressed. For example the "security" metrics element is expansive and distinctive enough to be included elsewhere, perhaps warranting its own design as the work of Quality of Security Service (QoSS) suggested by [Irvine 00]. The inclusion of the "combination" element into the performance metrics is also somewhat cumbersome and should be carefully utilized. Overall however this work provides an excellent initial endeavor into the quality of service and resource management issues that confront the contemporary, distributed environments of the present day. The quality of service event trace approach currently does not include security actions within its behavioral. However, this addition to the behavioral model is discussed as future work in chapter VII. Conclusion.

The work undertaken by [Kornegay 99] addresses system design as related to Very Large Scale Integration(VLSI) of architecture based upon quantitative quality of service aspects. Although this work is focused mainly upon the area of VLSI and CAD, portions of this research can be considered relevant to overall software system design.

This work examines efforts to reconcile the system design with quality of service optimizations. Here Kornegay defines quality of service as a function of the application properties. For example typical multimedia applications and many other applications place particular significance upon jitter, synchronization, and latency. There is a focus upon QoS-based Resource Allocation Model (Q-RAM) [Rajkumar 97] and Demand/Service (D/S) Curve [Sariowan 95] and combining these two approaches to resource management. Q-RAM focuses upon the concurrent multiple resource access (CPU, Disk, Network, Memory) while at the same time meeting the quality of service requirements such as timeliness, security, data quality, dependability, and reliable pack transmission. The object of Q-RAM is to enable the optimization of specific allocation of resources to an application under defined quality of service dimensions. These dimensions include: Sharable Resources; Applications needing resources; Application Utility Value; System Utility; and QoS Dimension requirements.

The Demand/Service (D/S) Curve is focused mainly upon network resources, and examines required demand and available services over a fixed packet switched network. Here quality of service is expressed as simple functions of this service curve and connection bursts. Using the fundamentals of this approach the quality of service is based upon specific time slots which store application computation demands latency and backlog can be calculated for demands and services and synchronized over a curve thus smoothing quality of service levels to functional proportions.

Interestingly [Kornegay 99] also states that “Behavioral synthesis suffered for many years lack of adequate benchmarks. Obviously, real-life QoS benchmarks are the mandatory prerequisite for CAD QoS tools. At the same time, there is a strong need for well documented design studies of complete systems which address QoS-sensitive applications.” The research presented in this document helps to address these issues.

The work of [Welch 99a] and [Welch 97]includes an approach to analyzing quality of service issues. In this work a dynamically operating quality of service monitor receives notification of violations and plans effective methods to recover from these violations. This work also notes a focus upon events, however the "event" of interest is an environment event such as situation assessment, monitor, and guide events that result from sensor input data and are not specifically quality of service events. Also this quality of service analysis work is established to operate within the Dynamic, Scalable, Dependable Real-Time Systems (Desiderata) resource management system [Welch 98], and its complexity may present issues when applying to other resource management approaches. This quality of service analysis approach is a good method for dynamically noting and recovering from quality of service violations, but it is not designed to be applied to the software engineering of programs with resource needs. In contrast the quality of service event trace analysis framework is designed to be readily applied to software engineering problems.

IV. EVENT TRACE ANALYSIS

This chapter presents the specific procedures and operations of applying the unique quality of service event trace analysis approach that has been developed by this dissertation research. This event trace analysis is focused on a targeted quality of service aware application program that has been pre-selected based upon a specific problem domain to substantiate the quality of service event trace process. A primary decision criterion for the quality of service application program selection is the distributed computing environment of the program. The program must accordingly include quality of service awareness which allow it to actively request and utilize resources. This decision process also includes, as a high priority, the evaluation of the program relevance to the existing DOD architectures, in this instance the AEGIS environment.

A. TARGET PROGRAM

The analysis of the selected target program is the specific case study for the validation of the general quality of service event trace approach. The proposed event trace procedures can be applied to most generic programs that require managed levels of quality of service. The Fast Failure Detection program was selected to be the targeted application based upon the problem domain prerequisites discussed in chapter II section B. The fast failure detection program has as its primary objective the efficient and prompt detection of node failures within group communication software as utilized within distributed mission critical systems of the AEGIS environment.

The specific event trace analysis effort has examined the quality of service events and related quality of service characteristics of this fast failure detection program within a distributed environment. This was accomplished through a low-level instrumentation of

the failure detection program sources. The fast failure detection program was designed and developed under the DARPA-ITO Quorum Integration, Testbed and Exploitation (Quite) project efforts. Leveraging from this environment the quality of service event trace analysis work has taken full advantage of this working testbed for the dissertation research efforts. Further discussion of the target program and its test environment can be found within chapter V.

B. OBJECTIVES OF THE EVENT TRACE ANALYSIS

A paramount objective of the quality of service event trace is to support risk reduction for programs that rely upon effective resource utilization. The quality of service event trace is a principal step in the development of the behavioral model that examines the quality of service performance of these programs. The development of the behavioral model is based upon specific quality of service actions and their respective attributes of interest as represented by event models. These behavioral components include:

- Quality of service request actions which requests resource reservation.
- Actions that focus upon the evaluation and negotiation of available resources to be applied to the originating resource request.
- Actions that denote proper utilization of the assigned resources.
- Actions responsible for the detection of any resource needs change within the application software.
- Actions focusing upon the re-negotiation based on increase or decrease of available and previously assigned resources.
- Re-allocation actions for specific resources by the resource controller.
- Resource type attribute associated with an quality of service action

- Event location attribute designating where a given quality of service action was executed.
- Attributes specifying the process type which executed the quality of service action.
- Quality of service event trace thread depth level attributes.
- Attributes denoting the size of the requested resource.

The rational for these modeling decisions are based upon the conviction that specific quality of service actions could directly influence resource deployment effectiveness and performance. This specific case study validates the quality of service event trace approach by illustrating the exact execution of quality of service specific actions that have a direct influence upon successful quality of service level attainment. This validation of the quality of service event trace approach is described in chapter VII.

C. EVENT TRACE GRANULARITY

The event trace granularity is based upon the predetermined program points that are associated to appropriate resource deployment and have a direct consequences upon overall quality of service behavior. These program points allow the event trace to concentrate on specific quality of service actions that are of interest to the overall quality of service behavior. The quality of service event trace program has specifically examined quality of service actions found within the failure detection software program that begins with the main function in the primary ffd.c module. Within this main function the traceable events include all quality of service related setup elements that are recorded as additions to the event trace path length. The failure detection program proceeds with the primary initialization sequence found in the “initialize” function call. This initialization of the fast failure detection program includes both quality of service specific elements as

well as non-related program initialization elements. For this reason only specific quality of service elements such as setting up thread & process urgency levels have been included in the event trace analysis. Again at each program point of the event trace the path length is also noted and recorded. Within the set_process_urgency function found in the urgency.c source file, the main quality of service resource set is negotiated through the create_rk_resource_set function call. The resource set is then created with the direct call to the resource kernel rk_resource_set_create. This resource kernel call returns a resource set handle in the form of a resource set name associated with the resource set specific details. In the event of failure a quality of service violation action is noted. Further references to the created resource set are accomplished through this handle access point. These quality of service events are noted as a continuation of the event trace path length and as quality of service trace level events and recorded as such.

Upon successful creation of the requested resource set within the main process, the failure detection program continues on to set up three distinct threads for processing various elements of the detection and monitoring software. These events of setting up the threads are also included and recorded in the event trace with respective path length notation.

The first of the three threads is the runcensustaker thread that establishes communication parameters and continues on to call the take_census function within the censustaker.c module. This thread then acts to determine node/host failure and is dependent upon dynamically collected census data. The event trace analysis program treats these initial startup actions as elementary path length events. The censustaker thread continues with a call to the set_thread_urgency function call found within the urgency.c module. A check is performed for the utilization of the urgency flag and this call is then forwarded to the set_rk_thread_urgency function that is also found within the urgency.c module. Next a test is performed to determine if the thread task requires explicit resources and forwarded to the set_rk_resource_set_urgency function. Within this function, also located within the urgency.c module, specific quality of service

requirements are established that identify the reserve types and quantities. These requirements are submitted to the resource kernel through the direct kernel call rk_cpu_reserve_create. Throughout the extent of this quality of service path the event trace program observes and records quality of service path length, resource requests, resource negotiation, resource assignment, resource denial, resource re-negotiation, and successful achievement of quality of service levels.

The second thread is the runheartbeat thread that also establishes initial communication parameters at its startup and proceeds to call the send_heartbeats function found within the heartbeat.c module. This module is responsible for performing a heartbeat generator action by sending periodic heartbeat messages to failure detection nodes within the distributed environment. The event trace analysis program logs these startup events as common path length events.

The last thread called the runlaggardreporter thread calls the check_laggards function that initializes itself to the communication group. This thread then continues by calling the check_for_laggards function within the hosts.c module. This module executes to dynamically check the host states for timeout values. Regarding the runlaggardreporter thread, the event trace analysis program is focused upon resource utilization as previously acquired through the main process.

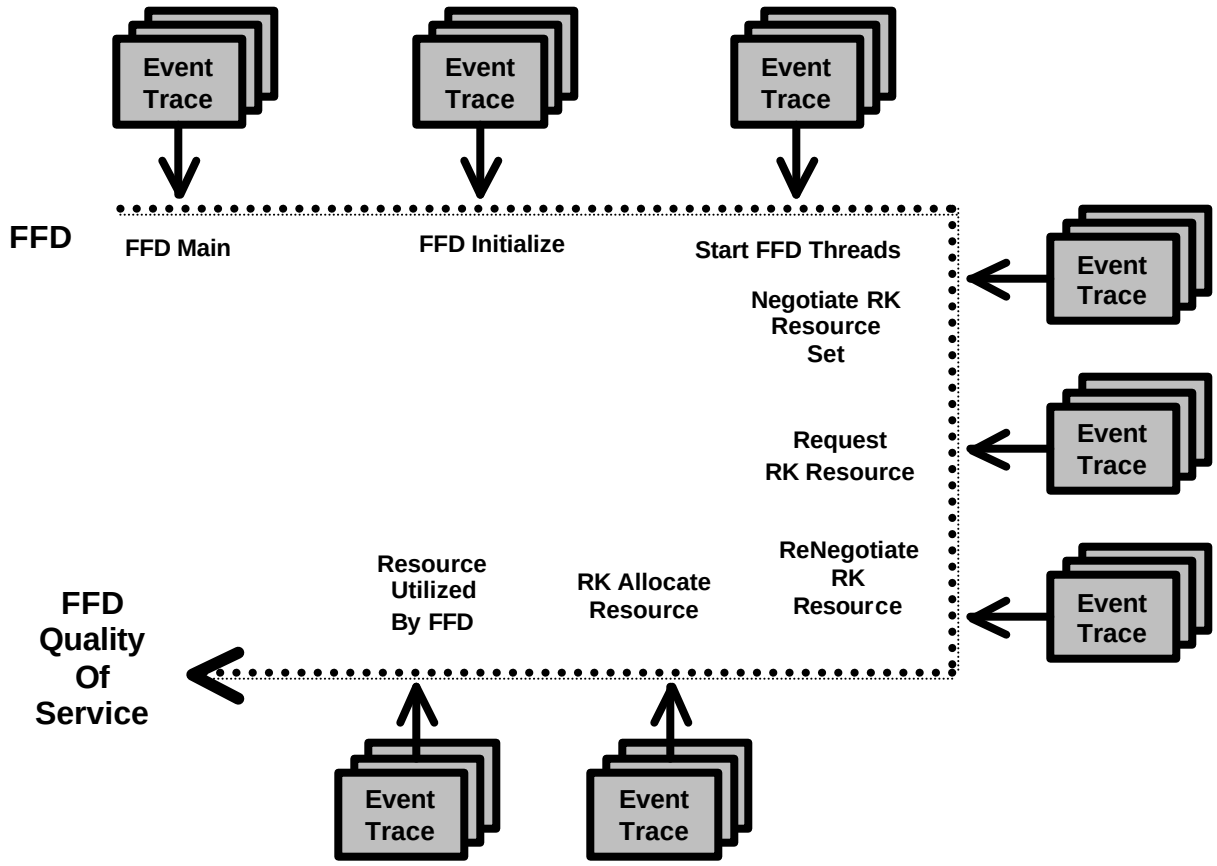

Figure 16. Event Trace Sequence

As illustrated in Figure 16 the event trace analysis effort essentially focuses upon the actions that are fundamental to resource utilization some of the typical program points where these actions occur are depicted here. This event trace includes all path length data as well as resource competition actions. During the event trace the failure detection program processes and threads compete for the resources that are being managed by the resource kernel. Two of the three threads asynchronously execute and specifically request distinct resources from the resource kernel. As stated earlier the third thread acquires resources from the parent process that have previously been allocated from the resource kernel. However, this resource competition also extends to other concurrently executing applications that request and in turn are provided with resources by the resource kernel. To intensify this competition a competing application is executed simultaneous to the failure detection program. This competing application requests exceptional quantities of

resources from the resource kernel. The explicit resources that have been allocated by the resource kernel for the competing application after processing its requests, are also noted and recorded within the event trace. In turn this previous resource allocation to the competing application has the potential to force re-negotiations of the resources being requested by the failure detection program threads and main processes. Any denial of the initial resource request and any re-negotiation actions for these resources by the failure detection processes and threads are recorded within the event trace.

THIS PAGE INTENTIONALLY LEFT BLANK

V. TESTBED ENVIRONMENT

As mentioned earlier the quality of service based event trace research effort has leveraged significantly from the DARPA Quorum program specifically the Integration Test & Exploitation project (Quite) testbed environment located at the SPAWAR Systems Center San Diego laboratory. This Quite testbed is composed of various heterogeneous software systems which employ assorted operating systems including Linux and Windows NT while encompassing diverse domains such as those illustrated in Figure 17[Lounsbury 99].

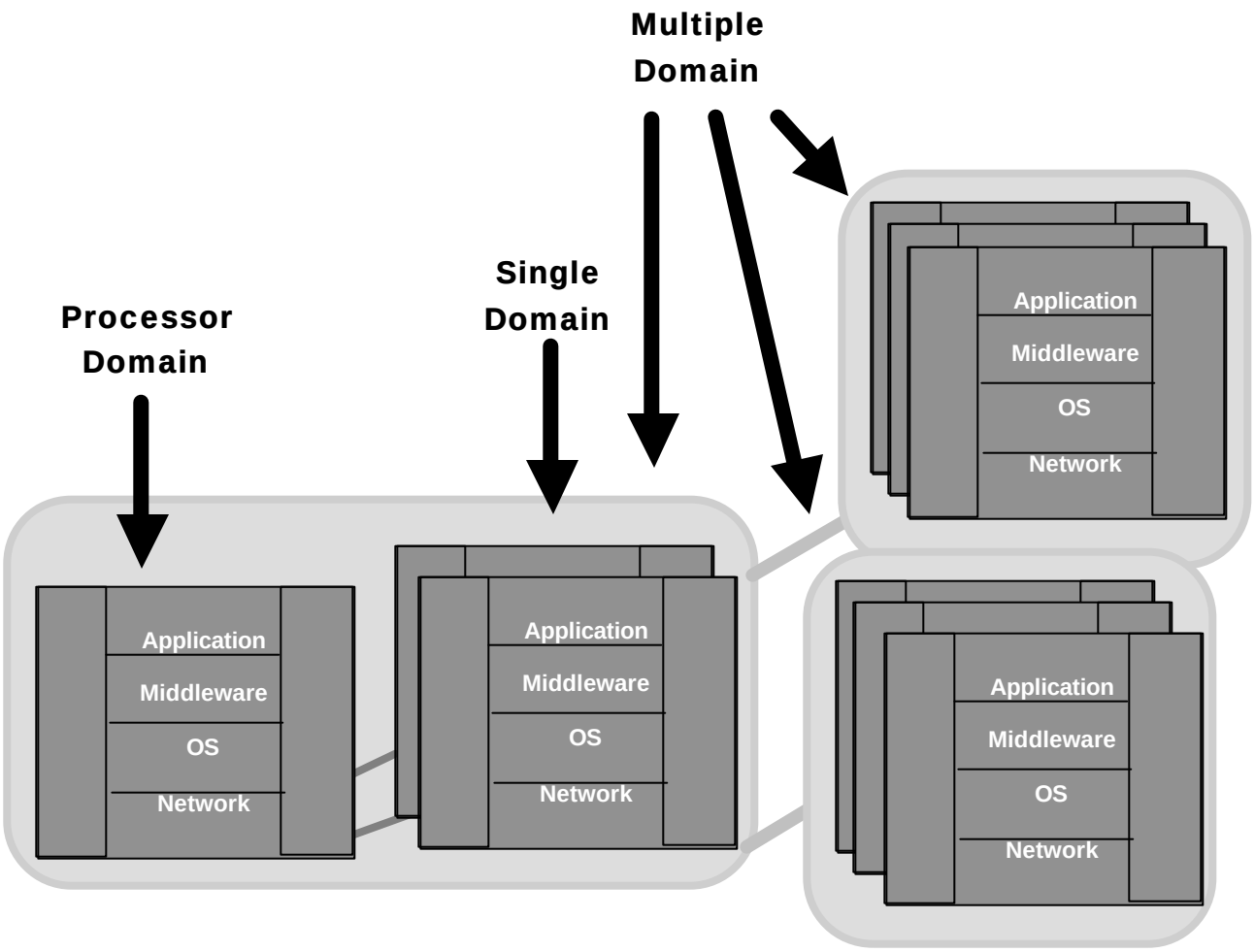

Figure 17. Domains Within The SSC-SD DARPA Testbed

This testbed also includes the fast failure detection[2] application program testing environment that employs linux as a base operating system and LinuxR/K as a resource kernel module. Additionally the Ensemble group communication software system & network tools are utilized within this environment.

The working hardware structure for this dissertation research effort includes five interconnected Intel Pentium based systems supplemented by network router and network hub devices. The testbed connectivity for this research that includes the fast failure detection software is illustrated in Figure 18. The network router RT1 as well as the network hub HB1 are both multiport and capable of up to 100Mb/s functionality using a 100 BaseT Ethernet. The Intel Pentium systems are all single processor architecture and contain 512MB of system memory. The Linux/RK systems are based upon linux Red Hat[3] version 7.0.

In the "Fast Failure Detection Test Environment" illustration, the system labeled Mon1 utilizes the linux resource kernel operating system and performs a heartbeat checking function. The system labeled Mon2 utilizes the WinNT operating system version 4.0 and essentially acts as a packet checker for system network traffic.

[2] A detailed discussion of the Fast Failure Detector program is located in chapter V.

[3] Red Hat is a registered trademark of Red Hat, Inc.

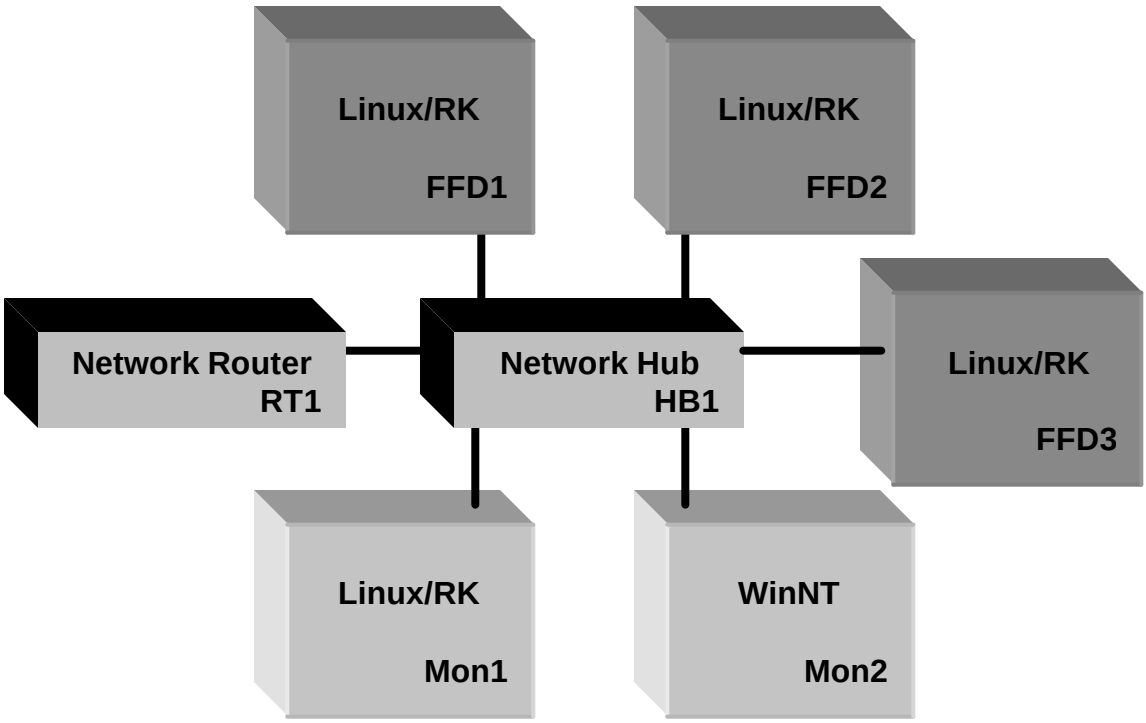

Figure 18. Fast Failure Detection Test Environment

A. LINUX

The linux operating system is a free preemptive Unix-type system that was initially developed by Linus Torvalds that included international contributions from numerous developers around the world. To understand the linux system we must reveal its origins that presents an interesting history of operating system development.

Pioneering operating systems have been improved as technology has progressed and these fundamental systems can be directly connected to the beginning of the Linux[4] operating system. The Linux system is also the result of numerous contributions from

[4] Linux is a trademark of Linus Torvalds

individuals as outlined in this short history segment by [Rusling, 99] “The roots of Linux can be traced back to the origins of Unix[5]. In 1969, Ken Thompson of the Research Group at Bell Laboratories began experimenting on a multi-user, multi-tasking operating system using an otherwise idle PDP-7. He was soon joined by Dennis Richie and the two of them, along with other members of the Research Group produced the early versions of Unix. Richie was strongly influenced by an earlier project, MULTICS and the name Unix is itself a pun on the name MULTICS. Early versions were written in assembly code, but the third version was rewritten in a new programming language, C. C was designed and written by Richie expressly as a programming language for writing operating systems. This rewrite allowed Unix to move onto the more powerful PDP-11/45 and 11/70 computers then being produced by DIGITAL. The rest, as they say, is history. Unix moved out of the laboratory and into mainstream computing and soon most major computer manufacturers were producing their own versions. Linux was the solution to a simple need. The only software that Linus Torvalds, Linux's author and principle maintainer was able to afford was Minix. Minix is a simple, Unix like, operating system widely used as a teaching aid. Linus was less than impressed with its features, his solution was to write his own software. He took Unix as his model as that was an operating system that he was familiar with in his day to day student life. He started with an Intel 386 based PC and started to write. Progress was rapid and, excited by this, Linus offered his efforts to other students via the emerging world wide computer networks, then mainly used by the academic community. Others saw the software and started contributing. Much of this new software was itself the solution to a problem that one of the contributors had. Before long, Linux had become an operating system. It is important to note that Linux contains no Unix code, it is a rewrite based on published POSIX standards. Linux is built with and uses a lot of the GNU (GNU's Not Unix) software produced by the Free Software Foundation in Cambridge, Massachusetts.”

The kernel can be considered the basis of the Linux operating system that also includes various system programs. User application programs also execute by using the

[5] UNIX is a registered trademark of X/Open

operating system. The kernel provides a separation of these programs from the direct employment of the hardware and other system resources. This centralized function of the kernel makes it an ideal candidate for a quality of service based resource allocation that is discussed in the next of this document.

As noted in [Stafford, 01] “The Linux kernel consists of several important parts: process management, memory management, hardware device drivers, filesystem drivers, network management, and various other bits and pieces…” This simplified breakdown is illustrated by Stafford in Figure 19 which diagrams some specific elements of the linux kernel.

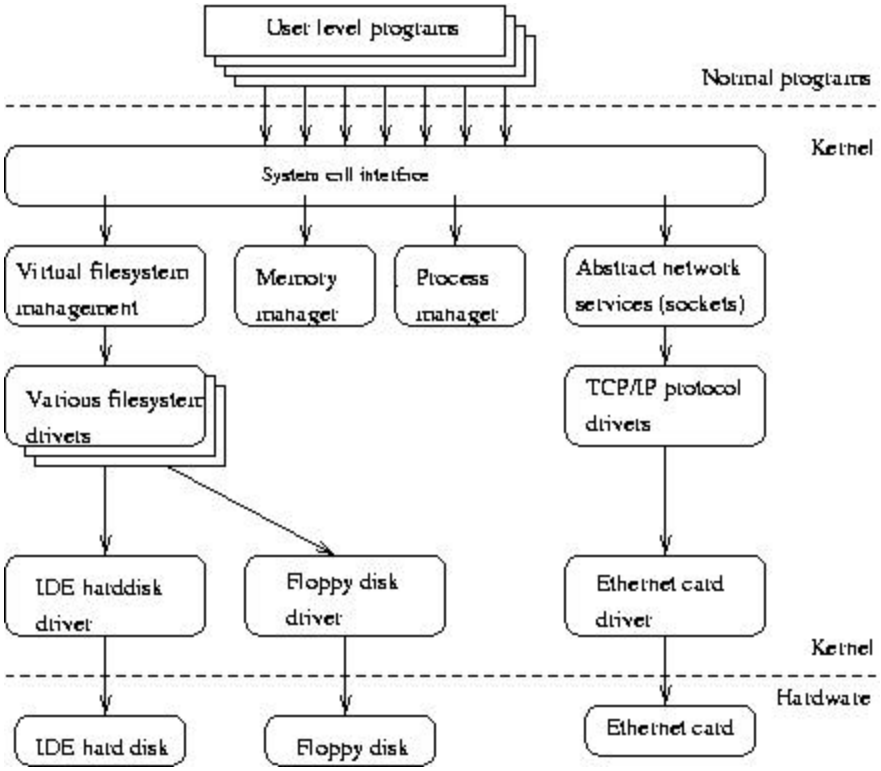

Figure 19. The Linux Kernel.

The point at which the Linux kernel directly interfaces to the system hardware includes numerous device driver software elements. These elements exhibit some commonality as described by [Stafford, 01] “At the lowest level, the kernel contains a hardware device driver for each kind of hardware it supports. Since the world is full of different kinds of hardware, the number of hardware device drivers is large. There are often many otherwise similar pieces of hardware that differ in how they are controlled by software. The similarities make it possible to have general classes of drivers that support similar operations; each member of the class has the same interface to the rest of the kernel but differs in what it needs to do to implement them. For example, all disk drivers look alike to the rest of the kernel, i.e., they all have operations like `initialise the drive', `read sector N', and `write sector N'.”

This section has presented only very brief description of the Linux operating system and is not intended to explore the extensive details of this system. However, for a more detailed discussion of the Linux operating system please consult the citations noted in the Reference chapter of this document including Rusling, David, A., *The Linux Kernel*, and Stafford, S., Wirzenius, L., Oja, J., *The Linux System Administrator's Guide, Version 0.7*. Also, for additional software use information regarding the Linux operating system please also see the Copyright And License section in the Appendix chapter of this document.

B. LINUX/RK

This segment provides specific information regarding the foundation of the quality of service system called Linux/RK. This system was developed by Carnegie Mellon University Real-time and Multimedia Systems Laboratory. The majority of this information describing Linux/RK is quoted from the CMU Real-time and Multimedia Systems Laboratory literature as noted in the reference section.

As stated in [CMU 97] "Linux/RK stands for Linux/Resource Kernel, which incorporates real-time extensions to the Linux kernel to support the abstractions of a resource kernel. A resource kernel is a real-time kernel (operating system) that provides timely, guaranteed and enforced access to system resources for applications." This controlled access to system resources allows Linux/RK to incorporate an efficient management and distribution of these resources which is based upon a quality of service model. The [CMU 97] document continues with a description of the capabilities of the resource-centric Linux/RK system "The resource kernel allows applications to specify only their resource demands leaving the kernel to satisfy those demands using hidden resource management schemes. This separation of resource specification from resource management allows OS-subsystem-specific customization by extending, optimizing or

even replacing resource management schemes. As a result, this resource-centric approach can be implemented with any of several different resource management schemes. The resource kernel gets its name from its resource-centricity and its ability to:

- Apply a uniform resource model for dynamic sharing of different resource types,
- Take resource usage specifications from applications,
- Guarantee resource allocations at admission time,
- Schedule contending activities on a resource based on a well-defined scheme

This linux resource kernel environment essentially links the two kernels together for the purpose of providing a method of effective system resource management for the requesting application processes. [Oikawa 98] also notes that "Linux/RK is an implementation of a resource kernel based upon the linux system." The Linux/RK system is integration composed of a linux kernel and portable resource kernel subsystem as illustrated in Figure 20. The Resource Kernel is a kernel module labeled the rk.o module that is loaded into Linux[6] by following a sequence referenced in [Komarinski 00]. The Linux/RK system utilized in the research discussed in this document is based upon the Linux Red Hat[7] version 6.27.0.

[6] Linux is written and distributed under the GNU General Public License Version 2, June 1991.

[7] Red Hat is a registered trademark of Red Hat, Inc.

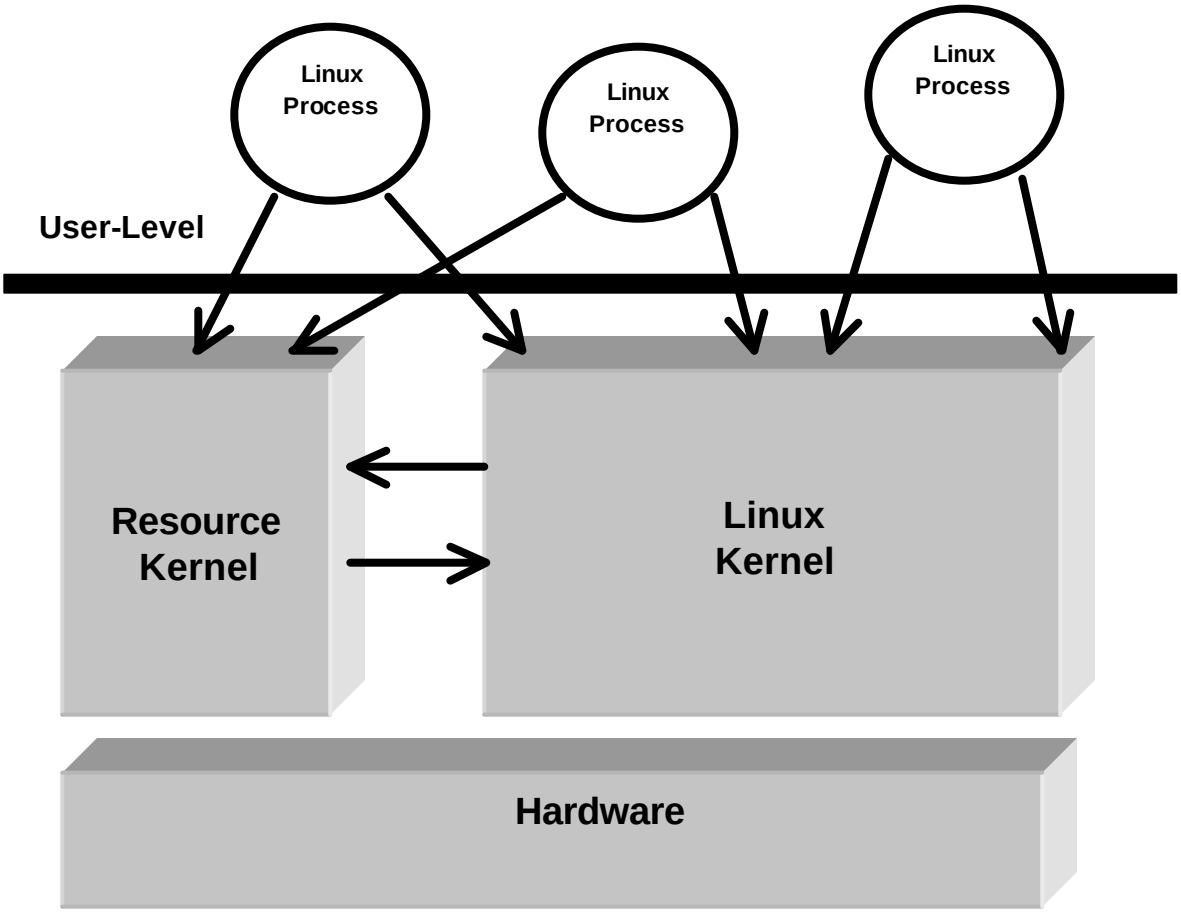

Figure 20. Linux Resource Kernel Architecture.

Once this resource kernel has been loaded the desired resources can be distributed to the requesting application programs in a way that facilitates efficient quality of service. Also noted in [Oikawa 98] "A QoS manager or an application itself can then optimize the system behavior by computing the best QoS obtained from the available resources."

The resource kernel reservation parameters depend upon the specific resource that is being reserved as summarized in [Oikawa 98] "A reserve can be time-multiplexed or dedicated. Temporal resources like CPU cycles, network bandwidth, and disk bandwidth are time-multiplexed, and spatial resources like memory pages are dedicated. A time-multiplexed resource has the following primary reserve parameters: C, D, T, where T

represents a recurrence period, C represents the processing time required within T, and D is the deadline within which the C units of processing time must be available within T." These parameters are illustrated in the [TimeSys 00] diagram found in Figure 21.

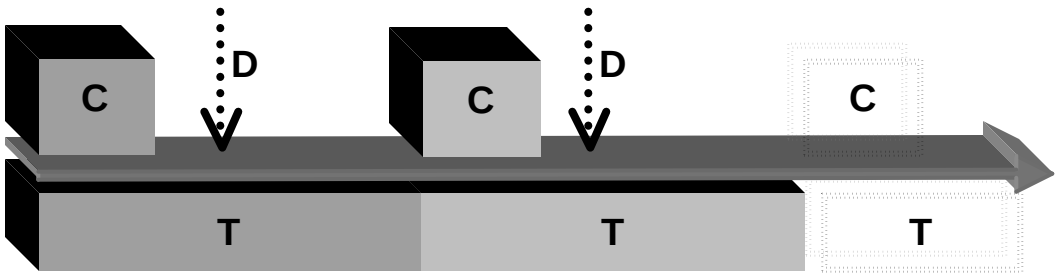

Figure 21. Resource Reservation Parameters.

The resource management model contains elements of resource depletion & replenishment. These terms are described by [Rajkumar 98] "When a reservation uses up its allocation of C within an interval of T, it is said to be depleted. A reservation that is not depleted is said to be an undepleted reservation. At the end of the current interval T, the reservation will obtain a new quota of C and is said to be replenished." As stated in [Oikawa 98] "In our resource management model, the behavior of a reserve between depletion and replenishment can take one of three forms:

- Hard Reserves: Will not be scheduled on depletion until they are replenished.
- Firm Reserves: Scheduled for execution on depletion only if no other undepleted reserve of unreserved resources users can be scheduled.
- Soft Reserves: Can be scheduled for execution on depletion along with other unreserved resource use and depleted reservations."

This section is not intended to provide the reader with an exhaustive understanding of the Linux/RK system but instead provides only a very brief

introduction. For those desiring to acquire an expanded understanding of the Linux/RK system please consult the citations noted in the Reference chapter of this document including Oikawa, S., Rajkumar, R., *Linux/RK: A Portable Resource Kernel in Linux*, Rajkumar, R., Lee, C., Lehoczky, J., Siewiorek, D., *A resource allocation model for QoS management*, and Rajkumar, R., Juvva, K., Molano, A., Oikawa, S., *Resource Kernels: A Resource-Centric Approach to Real-Time and Multimedia*. Additionally, for software use information regarding the Linux/RK system please also see the Copyright And License section in the appendix chapter of this document.

C. ENSEMBLE

The Ensemble[8] software program is a group communication software that has been developed at Cornell University Computer Science Department. The program design of the Ensemble system was partially leveraged from other previously developed communication software that was also developed at Cornell University. The Ensemble software is basically a third generation of these communication programs that include the Isis and Horus programs.

The Ensemble system includes various protocol layers as illustrated in Figure 22 from [Birman, 00] which reveals some of these as micro-protocols. This architecture is further described by Birman as "The basic idea underlying both (Ensemble & Horus) projects is to support group communication using a single generic architectural framework within which the basic group communication interfaces are treated separately from their implementation. One can then plug in an implementation matching the specific needs of the application. To maximize flexibility, each group end-point instantiates a stack of what we call micro-protocols. The developer arranges for the stack used in support of a given group to provide precisely the properties desired from the group. Each

[8] Ensemble was written and copyrighted in 1996 by Cornell University.

micro-protocol layer handles some small aspect of these guarantees. … Each process in a process group is supported by an underlying protocol stack; the stacks for the various members are identical, but the stacks used in different groups might be very different from one-another."

The typical utilization of Ensemble includes the employment of communication primitives and network tools as noted in [Birman 97] "Users of Ensemble treat it as a collection of tools and communication primitives." For the purposes of this dissertation research the Hot-C interface, Gossip server, Group Communication, heartbeat, and synchronization functionality of the Ensemble system are utilized within the fast failure detection program.

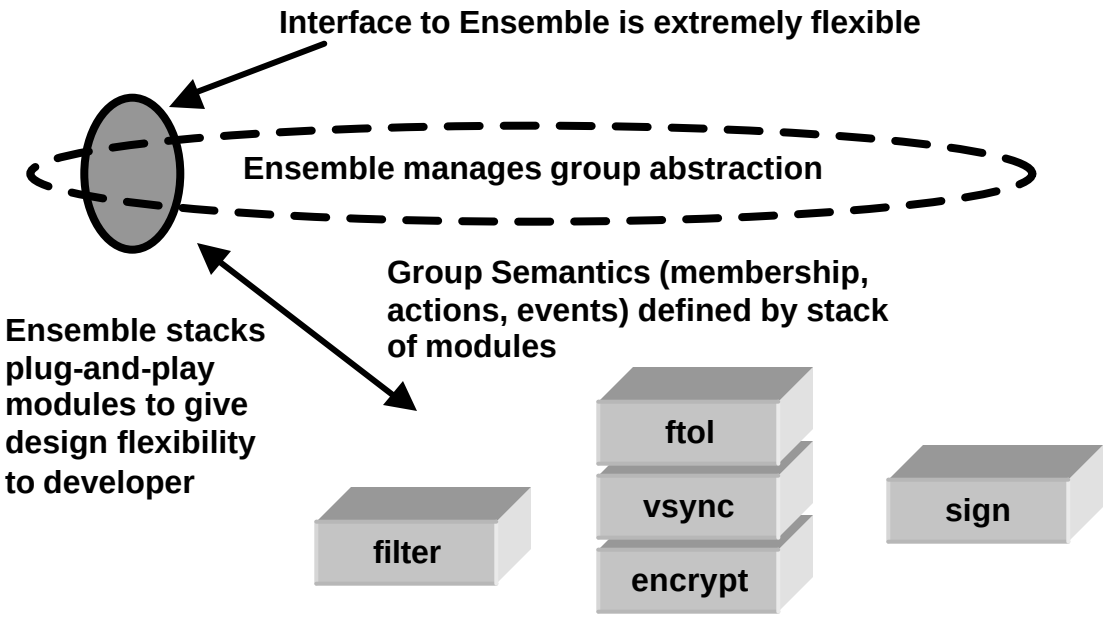

Figure 22. Layered Microprotocols In Ensemble.

This section on Ensemble is intended to provide a very brief description of the Ensemble system and is not designed to furnish the reader with a comprehensive discussion on this topic. To explore a more detailed discussion of Ensemble please consult the citations that can be found in the Reference chapter of this document including Birman, K., Vogels, W., Guo, K., Hayden, M., Hickey, T., Friedman, R., Maffeis, S., VanRenesse, R., Vaysburd, A., *Moving the Ensemble Groupware System to Windows NT and Wolfpack*, and Birman, K., et al., *The Horus and Ensemble Projects: Accomplishments and Limitations*. For further specific Ensemble program use information please also see the Copyright And License section in the Appendix chapter of this document.

D. FAST FAILURE DETECTOR

This section discusses the techniques and methodology behind the specific application program that the quality of service event trace analysis is focused upon for this research effort. The event trace analysis effort has examined the quality of service events and related quality of service characteristics of the application program called the Fast Failure Detector. This fast failure detection program was designed and developed under the DARPA-ITO Quorum Integration, Testbed and Exploitation (Quite) project efforts. The failure detection software was created within the Quite project testbeds, tested, experimented upon, and has ultimately been implemented within the Naval Surface Warfare Center Hiper-D AEGIS testbed. The primary objective of this failure detection software program is to efficiently and promptly detect node failures within the communication groups utilized in distributed mission critical systems such as the AEGIS environment.

As noted in [Drummond 02] this program can be set up to take advantage of a resource management system based upon quality of service procedures or operate as a

simple non-quality of service application “The Fast Failure Detector can be built and executed on its own or it can be executed while taking advantage of facilities like Linux/RK and Ensemble group communication.” For the purpose of this event trace analysis research the Fast Failure Dector program has been implemented using both the Linux/RK kernel and the Ensemble group communication software. A diagram of the Fast Failure Dector program operations and the specific program elements are illustrated in Figure 23.

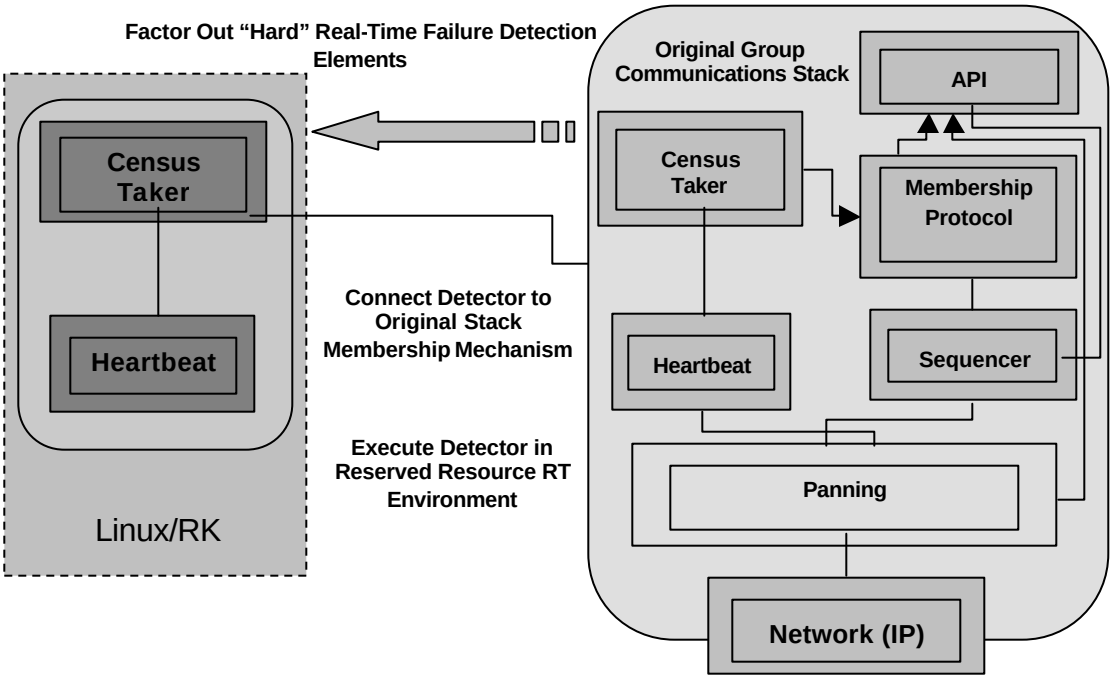

Figure 23. Fast Failure Detection Program.

As previously mentioned, prior to its implementation within the Naval Surface Warfare Center AEGIS environment the fast failure detection program was developed in the DARPA Quite project testbed and an experiment was constructed to examine the specific quality of service features and test its overall failure detection usefulness. As noted in [Drummond 02] "the purpose of this experiment is to investigate a proposed method for improving the behavior of systems that must meet simultaneous QoS requirements for both availability and timeliness." This primary investigation of the design of the fast failure detection experiment had projected that a typical system employing this failure detection software would have the capability of reliably detecting host node failures within sub-second time constraints.

The original goals of the fast failure detection experiment program are independent from the goals of the quality of service event trace that focuses exclusively upon system resource utilization efficiency. The fast failure detection goals are:

- Substantiate the assumption that the utilization of kernel based resource reservation results in a reduction of node failure recovery time.
- Validate the assumption that node failure detection time dominates failure recovery time in real-time FT.
- Establish component for inclusion into the technology transfer toolkit.

This section is not intended to provide the reader with an exhaustive understanding of the Fast Failure Detector program, but instead provides only a very concise description. For the readers desiring to know more about the design of this program or related program information please consult the citations noted in the Reference chapter of this document including Drummond, J., Wells, D., Rahman, M., *Detecting Failure Within Distributed Environments*, Additionally, portions of the program source code are included in the Appendix chapter, for software use information regarding the Fast Failure Detector program please also see the Copyright And License section also located in the Appendix chapter of this document.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. INVESTIGATION RESULTS

The resulting quality of service event trace examination data and related information are explained in this chapter that also includes a theoretical analysis of performance.

A. QUALITY OF SERVICE EVENT TYPES

For this application of the quality of service event trace analysis approach to the specific case of the selected targeted application the following distinct events and attributes have been recorded within each quality of service event trace execution. These events, their actions and attributes follow the event model developed in chapter II section B.

EVENT	ACTION	ATTRIBUTE
RES_NEG	Resource Negotiation	RES_TYP, PATH, LOC, LEVEL, PTYPE
REQ_RES	Resource Request	RES_TYP, PATH, LOC, LEVEL, PTYPE
RES_ASG	Resource Assignment	RES_TYP, PATH, LOC, LEVEL, PTYPE
PATH_LN	Quality of Service Program Point Traversal	PATH, LOC, LEVEL, PTYPE
QOS_VIO	Quality of Service Violation	RES_TYP, PATH, LOC, LEVEL, PTYPE
RES_RNG	Resource Re-Negotiation	RES_TYP, PATH, LOC, LEVEL, PTYPE, SIZE, PERIOD, DEADLINE.
QOS_LEV	Resource Appropriation	RES_TYP, PATH, LOC, LEVEL, PTYPE, SIZE, PERIOD, DEADLINE, USED
SYS_RES	System/Application Resource Reservation	RES_TYP, PATH, LOC, PTYPE, SIZE, PERIOD, DEADLINE, USED

Table 2. Quality of Service Events.

These specific event types that have been included within the quality of service event trace of the target failure detection program were chosen because they represent

distinct actions during program execution that have a direct influence resource utilization. These event types include attributes that are closely associated with and help describe these actions. The quality of service events occur asynchronously within the quality of service event trace and are informally structured with no overall strictly imposed ordering, however there is a partial ordering within each executing thread.

As shown in Table 2 the RES_NEG notation represents a quality of service action that transacts the negotiation of a resource set with the resource controller. The event attributes associated with this action include RES_TYPE, PATH, LOC, LEVEL, and PTYPE. The REQ_RES notation represents the quality of service action of requesting a resource from the resource controller. The event attributes associated with this event include RES_TYP, PATH, LOC, LEVEL, and PTYPE. The RES_ASG notation represents a quality of service action of the assignment of the requested resource by the resource controller. The event attributes associated with this event include RES_TYP, PATH, LOC, LEVEL, and PTYPE. The PATH_LN notation represents an action of traversing the quality of service path to another program point level. The event attributes associated with this event include PATH, LOC, LEVEL, AND PTYPE. The QOS_VIO notation represents the action of quality of service fault. This failure event has a direct causal relation to the preceding quality of service attempt action. The preceding events to the QOS_VIO action include: RES_TYP RES_NEG, RES_REQ, RES_RNG, RES_ASG attempts. The attributes that correspond to this event include RES_TYP, PATH, LOC, LEVEL, and PTYPE. The RES_RNG notation represents a quality of service action that denotes a re-negotiation of the resource size with the resource controller. The event attributes associated with this event include RES_TYP, PATH, LOC, LEVEL, PTYPE, SIZE, PERIOD, and DEADLINE. The QOS_LEV notation represents a quality of service action of the resource controller appropriating the requested resources. The event attributes associated with this event include RES_TYP, PATH, LOC, LEVEL, PTYPE, SIZE, PERIOD, and DEADLINE. The SYS_RES notation represents a quality of service action taken by a competing resource user that is not the target of the event trace. Which includes system programs and other application programs. The event attributes associated with this event include RES_TYP, PATH, LOC, PTYPE, SIZE, PERIOD, DEADLINE,

and USED. When the focus is directed only at the target program for evaluation the SYS_RES event simply represents a competing load application, to enable an evaluation of the target program under resource competition. For quality of service event trace with multiple target program analysis/evaluation this notation would not be used.

The RES_TYP notation represents the event trace attribute that denotes the type of resource reservation that is requested. This event attribute is not utilized during the analysis of this target failure detection program, as the sole resource that has been reserved by this program is the CPU resource. When it is used, the other possible resources that this attribute can represent include Disk, Network, and Memory. The TIME_TR notation represents the event trace attributes temporal aspects of the quality of service event trace. This event attribute is also not used during the analysis of this target program. When it is utilized this attribute provides a notion of timing for the quality of service event trace.

The PATH attribute references the total event trace quality of service path length that has been recorded during the application execution. This path element represents a simple integer value that is dynamically updated and recorded as the event trace proceeds. This integer value indicates the aggregate progression of the event trace.

The quality of service event trace attribute labeled LEVEL is similar to the Path element. However, this element reflects the specific process or thread execution path depth as it proceeds though the operations necessary to attain a specific level of quality of service. This element is also a simple integer that is dynamically updated and recorded as the process or thread executes. The LOC attribute references the specific processing location that the quality of service event trace is recording from within the specific process or thread. This attribute is a simple char type and includes FFDMAIN, FFDINIT, KSYSTEM, THREAD1, THREAD2, and THREAD3. The next attribute in the quality of service event trace output data is titled PTYPE. This attribute indicates the specific task

type that has been recorded. Two of the possible task types include process, and thread which are directly related to the failure detection application.

The SIZE attribute reflects the resource size being requested. The SIZE attribute is measured in resource units. The PERIOD attribute indicates the period that the resource is utilized within. The DEADLINE attribute relates to specific information utilized by Deadline Monotonic and Earliest deadline First scheduling policies. The USED attribute reflects the event of resources being allocated. The TOTAL element indicates the additive figure of all resources that have been allocated. The AVAL element is representative of the total resource available as indicated by the resource kernel. This element is also measured in resource units.

The other possible task types in this case study of the quality of service event trace include fabricated competing application tasks such as RS1, RS2, and system processes such as DISK. The competing application is a simple CPU resource load program that requests large amounts(400.0 & 200.0 units) of this resource. Its sole purpose is to present the target program with a resource competitor for evaluation under load. The system process is produced by the resource kernel for continuos disk access and requests a nominal amount of CPU resource(0.299 units). The resource kernel also indicates a setback of minimum 90 resource units that cannot be allocated.

The results of the event trace and their attributes are reported under notational areas that include ETTYPE, PATH, LEVEL, LOC, PTYPE, SIZE, PERIOD, DEADLINE, TOTAL, USED, and AVAL. These focus areas were selected to provide illumination of the quality of service related actions performed within a target program execution and are recorded in the following tables.

B. QUALITY OF SERVICE EVENT TRACE

ETTYPE	PATH	LEVEL	LOC	PTYPE	SIZE	PERIOD	DEADLINE	TOTAL	USED	AVAL
PATH_LN	1	1	FFDMAIN	PROCES				0.000		701
PATH_LN	2	1	FFDINIT	PROCES				0.000		701
PATH_LN	3	2	FFDINIT	PROCES				0.000		701
RES_NEG	4	3	FFDINIT	PROCES				0.000		701
PATH_LN	5	1	THREAD1	THREAD				0.000		701
PATH_LN	6	1	THREAD2	THREAD				0.000		701
PATH_LN	7	2	THREAD1	THREAD				0.000		701
PATH_LN	8	3	THREAD1	THREAD				0.000		701
PATH_LN	9	4	THREAD1	THREAD				0.000		701
PATH_LN	10	5	THREAD1	THREAD				0.000		701
PATH_LN	11	6	THREAD1	THREAD				0.000		701
REQ_RES	12	7	THREAD1	THREAD				0.000		701
QOS_LEV	13	8	THREAD1	THREAD	100.000	400.000	400.000	100.000	100.000	601
RES_ASG	14	9	THREAD1	THREAD				100.000		601
SYS_RES	15		KSYSTEM	DISK	0.299	28.506	28.506	100.299	0.299	600
RES_ASG	16		KSYSTEM	DISK				100.299		600
REQ_RES	17	10	THREAD1	THREAD				100.299		600
QOS_LEV	18	11	THREAD1	THREAD	100.000	400.000	400.000	200.299	100.000	500
RES_ASG	19	12	THREAD1	THREAD				200.299		500
PATH_LN	20	1	THREAD3	THREAD				200.299		500
PATH_LN	21	2	THREAD2	THREAD				200.299		500
PATH_LN	22	3	THREAD2	THREAD				200.299		500
PATH_LN	23	4	THREAD2	THREAD				200.299		500
PATH_LN	24	5	THREAD2	THREAD				200.299		500
REQ_RES	25	6	THREAD2	THREAD				200.299		500
QOS_LEV	26	7	THREAD2	THREAD	100.000	400.000	400.000	300.299	100.000	400
RES_ASG	27	8	THREAD2	THREAD				300.299		400
PATH_LN	28	2	THREAD3	THREAD				300.299		400
PATH_LN	29	3	THREAD3	THREAD				300.299		400
PATH_LN	30	4	THREAD3	THREAD				300.299		400
PATH_LN	31	5	THREAD3	THREAD				300.299		400
PATH_LN	32	6	THREAD3	THREAD				300.299		400

Table 3. Event Trace Data-1

The data shown in Table 3 illustrates the quality of service event trace behavioral analysis approach applied to the targeted failure detection application with minimal competition for quality of service resources. The quality of service event trace begins at the FDDMAIN location and proceeds through the FFDINIT locations. The available resource counter indicates that 701 resource units are open for allocation. These event types are simple notations of the path length, and the tasks are failure detection application processes. These processes request no specific resources during the initialization phases. However, at path 4 and the process event trace depth of level 3 in location FFDINIT process a direct call to the resource kernel negotiates a resource set for the failure detection application.

The quality of service event trace proceeds as the failure detection application spawns three threads. The event trace recorded these threads asynchronously, as the threads are asynchronous in their execution. At event trace path length 5 the first thread THREAD1 begins its initialization procedures. This initialization continues until path 12 and the thread event trace depth level 7. At this point the thread submits a quality of service request for resources from the resource kernel as noted by the quality of service event trace type of REQ_RES. The request is granted for the requested 100 units, and the quality of service event type is noted as QOS_LEV. The event trace records this amount as it is added to the cumulative total and the available resource amount is decreased by the granted units. The next quality of service event is recorded as a SYS_RES event trace type that is a system resource reservation, by the kernel task to provide for disk access. This system level resource reservation provides for the proper functioning of the Linux/RK kernel and is external to the target application and the event trace. The reservation units of 0.299 are recorded and added to the cumulative total of reserved resources, the same quantity of units are also subtracted from the available resource total. The resources are then formally assigned. The THREAD1 task again requests additional 100 units as noted by the event trace type REQ_RES at path 17 and thread event trace depth level 10. The request is again granted for the requested 100 units with a quality of service event type is noted as QOS_LEV. The event trace records this amount as it is also added to the cumulative total and the available resource amount is decreased by these 100 units. The formal assignment is also recorded. The THREAD2 task reaches the point in its processing, at path 25 and thread event trace depth level 6, where 100 resource units are needed. This action is noted and recorded in the quality of service event trace as a REQ_RES type event. The request is again granted for the requested 100 units at event trace depth 7. This is recorded as quality of service event QOS_LEV. The event trace also records this amount and it is added to the cumulative total and the available resource amount is decreased by 100 units. The formal assignment is also recorded. The remainder of the quality of service event trace encounters only simple path length type events.

The events of interest from the results of table 3 demonstrate a typical pattern of success behaviors composed of {RN}, and {RQ, QL, RA}. This overall quality of service behavior indicates effective allocation of the requested quality of service level.

The quality of service event trace illustrated in Table 3 reveals a simple resource distribution where ample resource units are available. This scenario provides limited insight into the application design with regard to quality of service issues. To achieve a greater understanding of the influence that an application program design could exert upon the achievement of sufficient quality of service levels we need to setup an environment where excessive competition for limited resources frequently occurs. When demand for resources exceeds the supply of these resources the corrective policy is the re-negotiation of the resource demand. This is provided for in the event model through the RES_RNG event.

This competition for resources must be based upon the total system resources available. These resources are distributed by the linux resource kernel as described in the earlier chapter within this document titled Testbed Environment. A competing application program has been included in this next quality of service event trace examination. The main purpose of the resource competition program is to essentially request large amounts of resources. This competition for limited available resources forces numerous re-negotiations for requested resources by the fast failure detection program.

ETTYPE	PATH	LEVEL	LOC	PTYPE	COMPUTE	PERIOD	DEADLINE	TOTAL	USED	AVAL
PATH_LN	1	1	FFDMAIN	PROCES				0.000		701
PATH_LN	2	1	FFDINIT	PROCES				0.000		701
PATH_LN	3	2	FFDINIT	PROCES				0.000		701
RES_NEG	4	3	FFDINIT	PROCES				0.000		701
PATH_LN	5	1	THREAD1	THREAD				0.000		701
PATH_LN	6	1	THREAD2	THREAD				0.000		701
PATH_LN	7	2	THREAD1	THREAD				0.000		701
PATH_LN	8	3	THREAD1	THREAD				0.000		701
PATH_LN	9	4	THREAD1	THREAD				0.000		701
PATH_LN	10	5	THREAD1	THREAD				0.000		701
PATH_LN	11	6	THREAD1	THREAD				0.000		701
SYS_RES	12		KSYSTEM	RS2	400.000	1000.000	1000.000	400.000	400.000	301
RES_ASG	13		KSYSTEM	RS2				400.000		301
SYS_RES	14		KSYSTEM	RS1	200.000	800.000	800.000	600.000	200.000	101
RES_ASG	15		KSYSTEM	RS1				600.000		101
REQ_RES	16	7	THREAD1	THREAD				600.000		101
QOS_VIO	17	8	THREAD1	THREAD				600.000		101
RES_RNG	18	9	THREAD1	THREAD	6.000	50.000	50.000	600.000		101
QOS_LEV	19	10	THREAD1	THREAD	6.000	50.000	50.000	606.000	6.000	95
RES_ASG	20	11	THREAD1	THREAD				606.000		95
SYS_RES	21		KSYSTEM	DISK	0.299	28.506	28.506	606.299	0.299	94
RES_ASG	22		KSYSTEM	DISK				606.299		94
REQ_RES	23	12	THREAD1	THREAD				606.299		94
QOS_VIO	24	13	THREAD1	THREAD				606.299		94
RES_RNG	25	14	THREAD1	THREAD	6.000	50.000	50.000	606.299		94
RES_RNG	26	15	THREAD1	THREAD	4.000	25.000	25.000	606.299		94
QOS_LEV	27	16	THREAD1	THREAD	4.000	25.000	25.000	606.000	4.000	90
RES_ASG	28	17	THREAD1	THREAD				610.299		90
PATH_LN	29	2	THREAD2	THREAD				610.299		90
PATH_LN	30	3	THREAD2	THREAD				610.299		90
PATH_LN	31	4	THREAD2	THREAD				610.299		90
PATH_LN	32	5	THREAD2	THREAD				610.299		90
REQ_RES	33	6	THREAD2	THREAD				610.299		90
QOS_VIO	34	7	THREAD2	THREAD				610.299		90
RES_RNG	35	8	THREAD2	THREAD	6.000	50.000	50.000	610.299		90
RES_RNG	36	9	THREAD2	THREAD	4.000	25.000	25.000	610.299		90
RES_RNG	37	10	THREAD2	THREAD	2.000	12.000	12.000	610.299		90
QOS_VIO	38	11	THREAD2	THREAD				610.299		90
PATH_LN	39	1	THREAD3	THREAD				610.299		90

Table 4. Event Trace Data-2

The purpose of this scenario illustrates the quality of service event trace behavioral analysis approach applied to the targeted failure detection application with maximum competition for quality of service resources. As illustrated within the Table 4 the same initialization and startup processes are recorded in this event trace as witnessed in the previous scenario. The quality of service event trace begins at the FDDMAIN location and continues through the FFDINIT locations. The total quality of service event trace also indicates that the available resources include 701 resource units that can be reserved. Again at the processes at path 4 and event trace depth of level 3 a direct call to

the resource kernel negotiates a resource set for the failure detection application. The failure detection application next starts up the three threads.

At path 16 and thread event trace depth of level 7 THREAD1 has concluded its initialization process and sends a call to the resource kernel requesting resources, as noted by the event trace type REQ_RES. However, the competing application has spawned tasks RS2 and RS1 that have already reserved 400 and 200 resource units respectively. These previous reservations reveal that the available resource level has been significantly reduced and the resource kernel denies the THREAD1 task the requested resources. This condition equates to a classic quality of service violation and the event type of QOS_VIO is recorded by the event trace when THREAD1 is denied its requested resources. The THREAD1 task is then forced into a re-negotiation condition for the duration of 1 event trace depth level. This thread is eventually granted the re-negotiated resource units that have been reduced significantly from the originally requested amount. These resources are then formally assigned and THREAD1 again requests its second reservation at path 23 and thread event trace depth level 12. Once again the resources are not available, in addition to the disk task has also reserved 0.299 units, and a quality of service violation is recorded with a QOS_VIO event type by the event trace. This condition forces a second re-negotiation for resources from the resource kernel as noted by the RES_RNG event type. The re-negotiation process is successful and a reduced number of resource units are allocated to the THREAD1 task by the resource kernel. These resource units are then also formally assigned to the THREAD1 task and the resource unit totals are recalculated.

The quality of service event trace process continues by logging simple path length events until the event trace path reaches the length of 33 and thread depth level 6. At this point the THREAD2 task submits a request for resources from the resource kernel as noted by the event trace event type of REQ_RES. The available resources maintained by the resource kernel have been reduced significantly and the request is denied, as the resource kernel must retain a setback of reserves hence approximately 90 units remain. This denial condition is recorded by the quality of service event trace as a QOS_VIO

event type. The THREAD2 task attempts to re-negotiate the resource amount with the resource kernel as noted by the RES_RNG event type. However, this re-negotiation process completely fails and the resource kernel eventually cannot fulfill the THREAD2 task requested resources. This action again results in a QOS_VIO event type that is recorded by the quality of service event trace.

The events of interest from the results of table 4 illustrate various quality of service behaviors.

- Success behavior
    - The events of interest from the results of Data-2 demonstrate a typical success behavior composed of {RN}.
- Potential failure behavior
    - The events of interest from the results of Data-2 illustrate a progressive pattern of potential for failure that concludes with a quality of service violation and final failure.
    - This progression can be see in patterns composed of {RQ, QV, RR, QL, RA}, {RQ, QV, RR, RR, QL, RA}, {RQ, QV, RR, RR, RR, QV}.
- Failure Behavior
    - The behavioral model indicates quality of service failure in the event trace results data-2. The failure occurrence shows the events {RQ, QV, RR, RR, RR, QV}.

The program point of quality of service failure can be isolated through an examination of the event trace results.

- Failure Event Attributes
    - This examination includes a retrace of the execution path to the specified depth level of the distinct(named) thread/process, and by isolation of the

event attributes associated with the failure event quality of service violation QV = {DP, TP, LC, PA, RT}. Where DP= 7-11, TP=THREAD, LC=THREAD2, PA=34-38, RT=CPU.

ETTYPE	PATH	LEVEL	LOC	PTYPE	SIZE	PERIOD	DEADLINE	TOTAL	AVAIL
QOS_VIO	34	7	THREAD2	THREAD				610.299	90
RES_RNG	35	8	THREAD2	THREAD	6.000	50.000	50.000	610.299	90
RES_RNG	36	9	THREAD2	THREAD	4.000	25.000	25.000	610.299	90
RES_RNG	37	10	THREAD2	THREAD	2.000	12.000	12.000	610.299	90
QOS_VIO	38	11	THREAD2	THREAD				610.299	90

A progressive pattern of potential for failure that concludes with a quality of service violation and final failure. This progression can be see in patterns composed of {RQ, QV, RR, QL, RA}, {RQ, QV, RR, RR, QL, RA}, {RQ, RR, RR, RR, QV}.

It is interesting to note in performance analysis that this complete denial of resources could have a catastrophic consequence for the requesting thread or process. This result could also translate into an uncertain outcome for the total program execution resulting in total program failure. This instance of a potential for total program failure was evident in the case study of the fast failure detection program. During the quality of service event trace analysis execution that included competition for resources that produced the data found in Table 4, the fast failure detection program exhibited a total collapse of the THREAD2 task. This failure of the specific thread to reserve necessary resources from the resource kernel in turn resulted in an abort of the program.

THIS PAGE INTENTIONALLY LEFT BLANK

VII. CONCLUSION

This chapter provides a summary of the completed dissertation work that has been presented in this document. These concluding elements contain a restatement of the initial research work and the resulting findings. Pending issues from this work related to command & control, distributed computing, and collaborative processing are also discussed. Additionally the scope of future work is also discussed in the final section of this chapter.

A. ORIGINAL OBJECTIVES

As stated in the Introduction section of this document the long-term contributory goal of this work is to aid in the development of an approach for designing distributed command & control systems with regard to efficient resource utilization. The specific goals and objectives of this research include:

- Development of practical event models based upon quality of service actions.
- Development of representative high-level quality of service behavioral models focused upon domain specific areas within a distributed environment, which benefit from quality of service treatment. The domain specific targeted system is the command and decision element found within the AEGIS architecture.
- Construction of a framework for applying the quality of service behavioral analysis to applications within the distributed command & control environment.

- Application of this framework to perform an effective quality of service event trace upon a targeted application within the command & control domain to isolate potential failure points.
- Performance of an appraisal of the influence that quality of service level requirements specification can exert upon software engineering within distributed environments. There will be no specific metrics involved in this simple e.g. good/bad appraisal that identifies specific quality of service effects, records the observed behavior and describes the potential influence.

The completion of these objectives have necessitated the creation of a system and processes that have combined the results of a quality of service based event trace performed upon a typical distributed system application and the specific resource utilization/management needs of applications within this distributed system. This event trace has been completed to analyze quality of service efficiency levels. The result of this effort has produced a series of quality of service characteristics that can be applied to a generalized system design as well as to other areas that are directly related to the goals and objectives described here. The discussion of these resulting findings and their relationship to the goals and objectives can be found in the next section of this chapter.

B. RESULTING FINDINGS

The findings resulting from the quality of service event trace analysis include elements that have directly addressed the original goals and objectives of this dissertation research effort. This section focuses upon the concluding discussion for the specific application of the quality of service event trace results to command & control, distributed computing, and software engineering areas. Suggested explicit deployment of the quality of service analysis framework into these areas is discussed as well as the resolution of the original goals and objectives.

The first goal of this research effort has been to develop practical event models based upon quality of service actions. The event model has been constructed from specific quality of service based actions and the relevant attributes that describe these actions. A total of eight actions of interest and their respective attributes have been used in the development of the event model. These events include: resource request event ‘RQ, resource negotiation event ‘RN, resource assign event ‘RA, quality of service event ‘QV, resource re-negotiation event ‘RR, quality of service level event ‘QL, system reservation event ‘SR, and path length event ‘PL. The event model has been applied to the event trace for the purpose of constructing the quality of service behavior as illustrated in . A more detailed discussion of the event model composition can be found in chapter II section B.

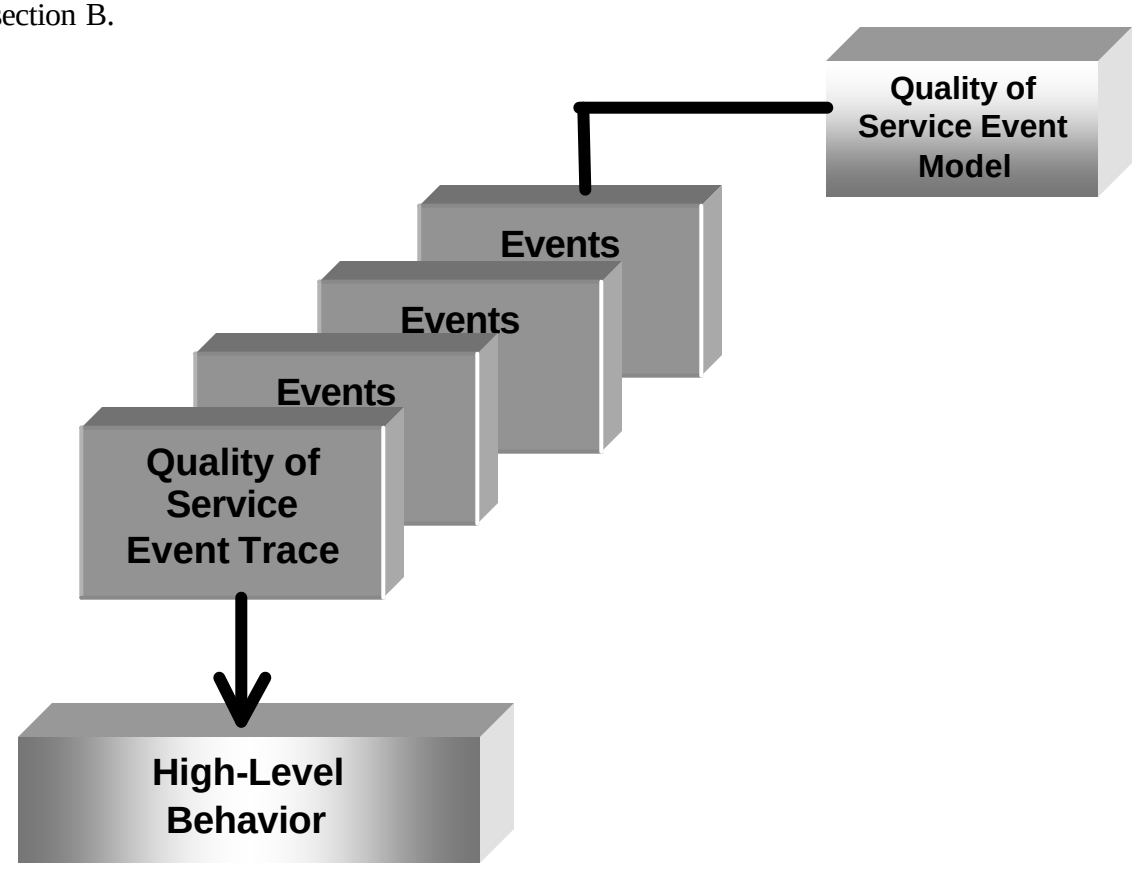

Figure 24. Applying The Event Model.

The second goal of this dissertation research is the development of representative high-level quality of service behavioral models that focus upon domain specific areas of a distributed environment to characterize quality of service treatment of resources within this environment. The behavioral model has been developed and is derived from the eight event models. The behavioral model is composed of various quality of service events that include: resource request event ‘RQ, resource negotiation event ‘RN, resource assign event ‘RA, quality of service event ‘QV, resource re-negotiation event ‘RR, quality of service level event ‘QL, system reservation event ‘SR, and path length event ‘PL. Potential failure behaviors are comprised of the following sets of events: {RN, QV}, {RQ, QV}, {RQ, QV, RR, RR, RR, QV}, {RR, QV}, and {RA, QV}. Typical success behaviors are composed of sets of events that include: {RN}, {RQ, QL, RA}, {RR, QL, RA}, and {RA}. A more detailed discussion of the behavior model and its composition can be found in chapter II section B.

The third goal of this dissertation research is the formation of a framework that will enable employment of the quality of service behavioral analysis on applications within distributed command & control envrionments. The framework has been created from the following approach sequence:

- Select a typical distributed command & control environment.
- Isolate a specific element of a typical command & control environment, and select a target program.
- Develop operative models of execution pathways (quality of service & resource management specific statement execution) within the target application in this specific element based upon identifiable details such as resources requested, resources utilized, resources available, etc.
- Initiate the development of a working model of program behavior based upon quality of service factors. This is accomplished by producing abstractions of events that are fundamental to specific quality of service actions performed during typical program execution.

- Identify quality of service specific application program points that directly relate to appropriate resource utilization as illustrated in Figure 5. Such elements have direct consequences upon quality of service behavior.
- Instrument the targeted program based upon these previously identified specific quality of service program points.
- Perform quality of service event trace analysis upon the executing targeted program using this instrumentation.

This quality of service analysis framework can be applied to the development of the quality of service behavior that is representative of the targeted program. The concepts used to develop this framework and a discussion of its characteristics can be found in chapter II section B.

The fourth goal of this dissertation research effort is to apply this quality of service event trace framework approach to a targeted program within the command & control domain to isolate potential quality of service failure points. To validate the event trace approach the analysis framework has been successfully applied to a pre-selected targeted program called the Fast Failure Detector as illustrated in Figure 25. The targeted Fast Failure Detector program has the task of efficiently and promptly discovering node failures within communication groups utilized in distributed mission critical systems. A more detailed discussion of the failure detection program software and its objectives can be found in chapter V section D.

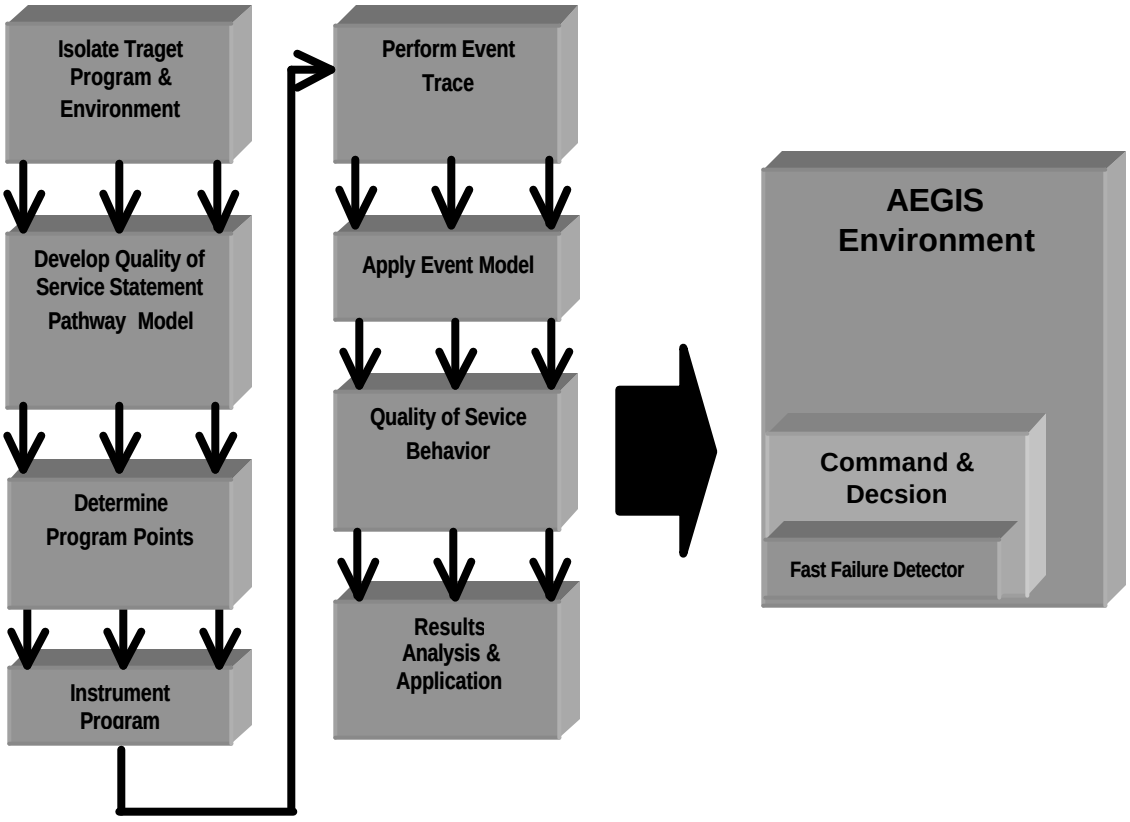

Figure 25. Employing The Quality Of Service Analysis Framework.

This quality of service event trace has directly examined the fast failure detection application that has been implemented within the command & decision element of an AEGIS testbed environment. The concluding results of this examination have produced accurate high-level quality of service behavior representations of the fast failure detection program. As previously declared in the Investigation Results chapter, this quality of service event trace analysis has shown the capacity to specifically reveal various potential failure points, potential resource re-negotiation inefficiencies. The exact failure points of the targeted program can be identified through an examination of the quality of service event trace results. As noted in the event trace data-2 table there are various QOS_VIO events through this output data. By performing a simple scan of this table the exact location of the potential for failure can be identified by the PATH, DEPTH, and LOC event attributes. The potential for resource re-negotiation inefficiencies is measured through examination of the re-negotiation process at the precise program points (PATH

& DEPTH) depicted by the RES_RNG event during the quality of service event trace. By analyzing the success or failures of the resource re-negotiation action, as indicated by an ensuing RES_ASG or QOS_VIO event, potential re-negotiation inefficiencies can be examined. All of these elements have a direct bearing upon the quality of service based resource management efficiency for the distributed command & control fast failure detection application program and environment.

The fifth goal of this dissertation research includes an appraisal of the influence that quality of service specification can exert upon the software engineering within distributed environments. This assessment of the quality of service specification has identified specific quality of service actions through examination of the behavior characteristics resulting from applying the event trace analysis framework to the targeted failure detection program and will describe the potential for influence to software design.

The observed behavior of multiple sequential resource re-negotiation actions was frequently noted in the event trace data-2 table. The resource re-negotiation action is representative of the subject threads attempt to correct a preceding quality of service violation. These simple actions of multiple sequential resource re-negotiation due to quality of service violations resulting from resource request actions and are indicative of a potential design deficiency in the area of quality of service efficient resource deployment. The required amount of a given resource that is needed for proper execution should be a critical design requirement. Neglecting this issue of quality of service level requirements when designing a program can lead to an unsuccessful resource re-negotiation event and can contribute to or cause total program failure as noted in the Investigation Results chapter.

Numerous quality of service violation actions that have also been observed during the event trace as can be seen in the event trace date-2 table. The quality of service violation action is an ordered event that is representative of a quality of service fault

behavior. This failure action has a causal relation to the preceding quality of service correlated attempt actions that include resource negotiation, resource request, resource re-negotiation, and resource assign. The event model that characterizes the quality of service violation event could consist of a resource request failure, or resource negotiation failure, or resource re-negotiation failure, or resource assigned failure actions. By direct examination of the behavior characteristics resulting from applying the event trace analysis it is determined that the quality of service violation actions are all representative of resource request failures. This failure event indicates that a potential design problem exists within the failure detection program and/or the resource controller. The quality of service violation event if not re-negotiated successfully can lead to unpredictable thread/process behavior for non-fatal violations and directly to thread/process and/or program failure in the case of fatal violations. The program design should include specific attention to the treatment of potential quality of service violations. This program design should also include a strategy for recovery from fatal quality of service violations. This condition could indicate a faulty or non-existent admission control module. The resource controller design should include recovery algorithms when admission control denies a request for resources. A possible remedy for this failure condition is the implementation of a priority based value system in conjunction with an effective admission control algorithm. The neglect of these issues when designing these software elements can result in potential program failure as illustrated in chapter VI.

This application of the quality of service analysis framework has focused precisely upon a specific case of the targeted failure detection program within a distributed command & control environment. This utilization of the quality of service event trace analysis can be applied to other programs within the distributed command & control domain, as well as many other DOD and commercial systems. This is accomplished by employing the quality analysis framework to the targeted program within these respective environments. These prospective targets of the quality of service event trace analysis would be within the development phases or could be legacy programs undergoing software engineering revisions. Regarding the long-term contributory goal of this dissertation research a framework has produced that can be

applied to software engineering areas that include, fine-tuning the quality of service elements within a prototyping design, correcting any quality of service inefficiencies or potential for failure in quality of service aware legacy programs as illustrated Figure 26. The efficient management process of correctly obtaining effective quality of service resource levels have also effectively been characterized by this dissertation research and can further be applied to this software engineering design and development process. This is accomplished by utilization of the quality of service event trace to enhance quality of service awareness that will in turn augment the creation of the program prototyping. In this way the quality of service event trace analysis can facilitate the creation of distributed command & control programs which have quality of service requirements.

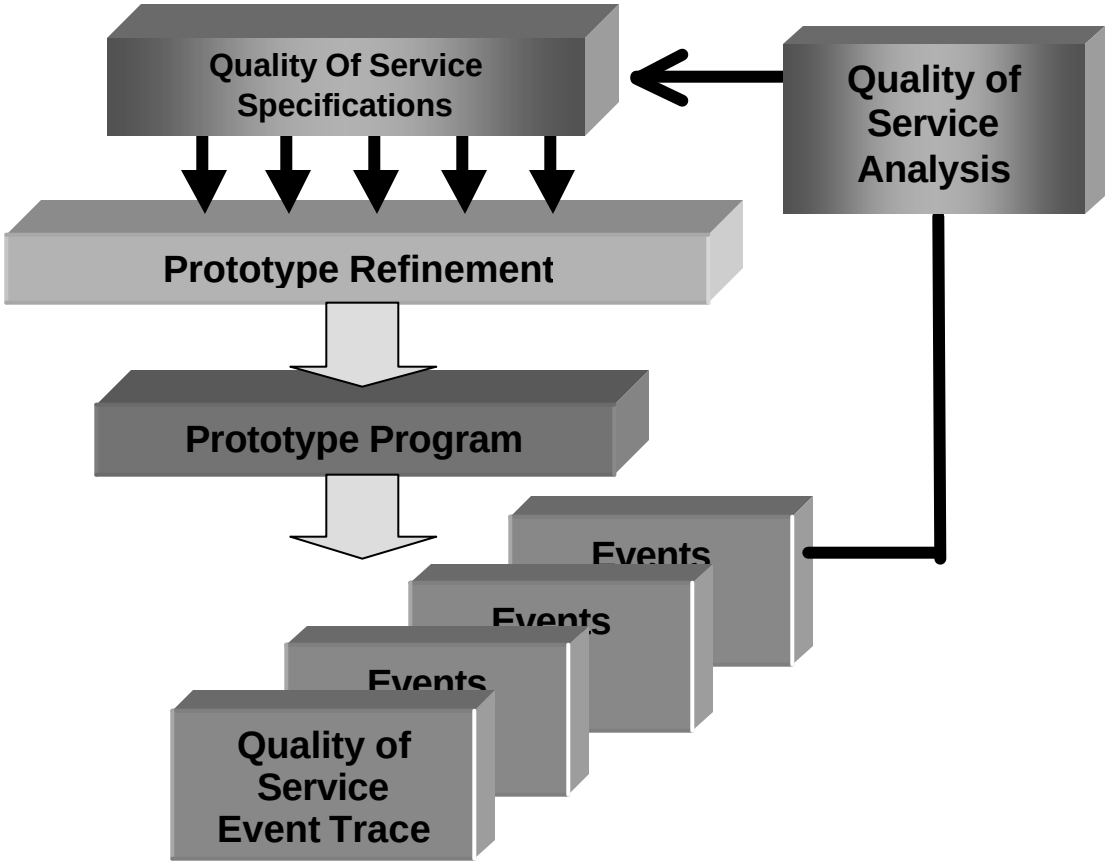

Figure 26. Quality Of Service Analysis.

The quality of service event trace efforts may also be effectively utilized in the implementation phase of the software engineering process, and also assist in refinement

of the overall requirements analysis for quality of service constraints as illustrated in Figure 27. During the implementation phase it is typically a complex task to ascertain the correct quality of service specifications that are to be utilized for a given distributed command & control program. For this method of employing the quality of service event trace analysis the results can be applied directly to the implementation of specific command & control applications. Potential failures or potential for quality of service errors can be revealed prior to deployment of any mission critical applications which exhibit a priority need to maintain significant quality of service levels. The instrumentation of the mission critical application program at quality of service specific program points will provide for the proper feedback during the execution of the pre-selected scenario that can be based upon worst/best case resource competition conditions. The application of the quality of service event trace may also include legacy programs as targets that have the capability for utilization of resource management controls and quality of service cognizance.

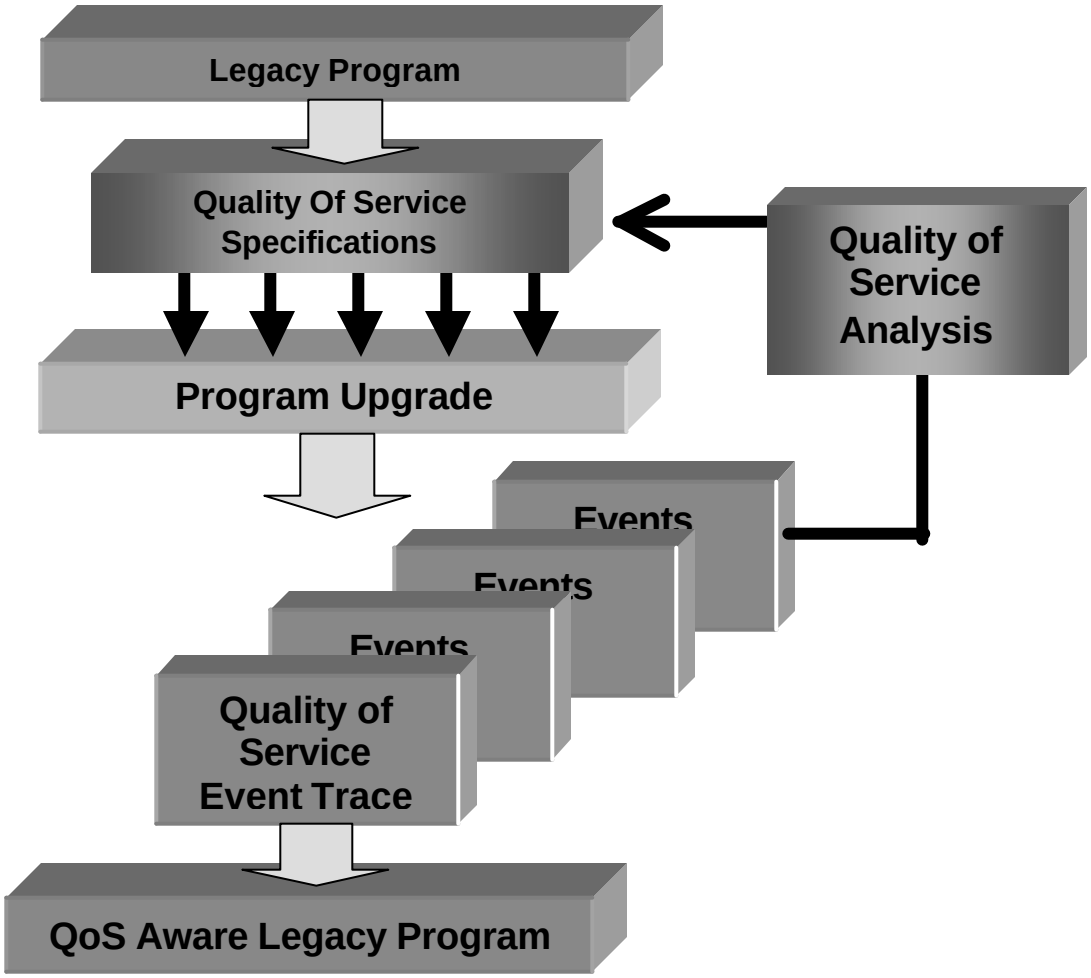

Figure 27. Legacy Program Implementation.

Effective quality of service issues such as adequate management of system resources within the distributed command & control environment such as CPU, disk, memory, and network bandwidth can be identified by the quality of service event trace as illustrated in Figure 28. Within this configuration the event trace is utilized to provide non-dynamic feedback data regarding the effective performance of the quality of service characteristics of the Linux/RK system. The resulting data from the quality of service event trace, as based upon a targeted application program, can be utilized to contribute to the evaluation of the quality of service system attributes. The event trace observes the overall system resource utilization, capacity, and availability, as well as the practical operation of the resource manager. Excessive indications of quality of service failure actions as noted by the event trace can be recorded in addition to any inordinate amounts

of events denoting re-negotiation for available resources. The resulting findings can be utilized in the specific effort to address the identification of a good method for the introducing quality of service attributes into the distributed environment. This endeavor assists in the accurate examination of the resource control module approach to introducing quality of service attributes such as resource management into the distributed environment.

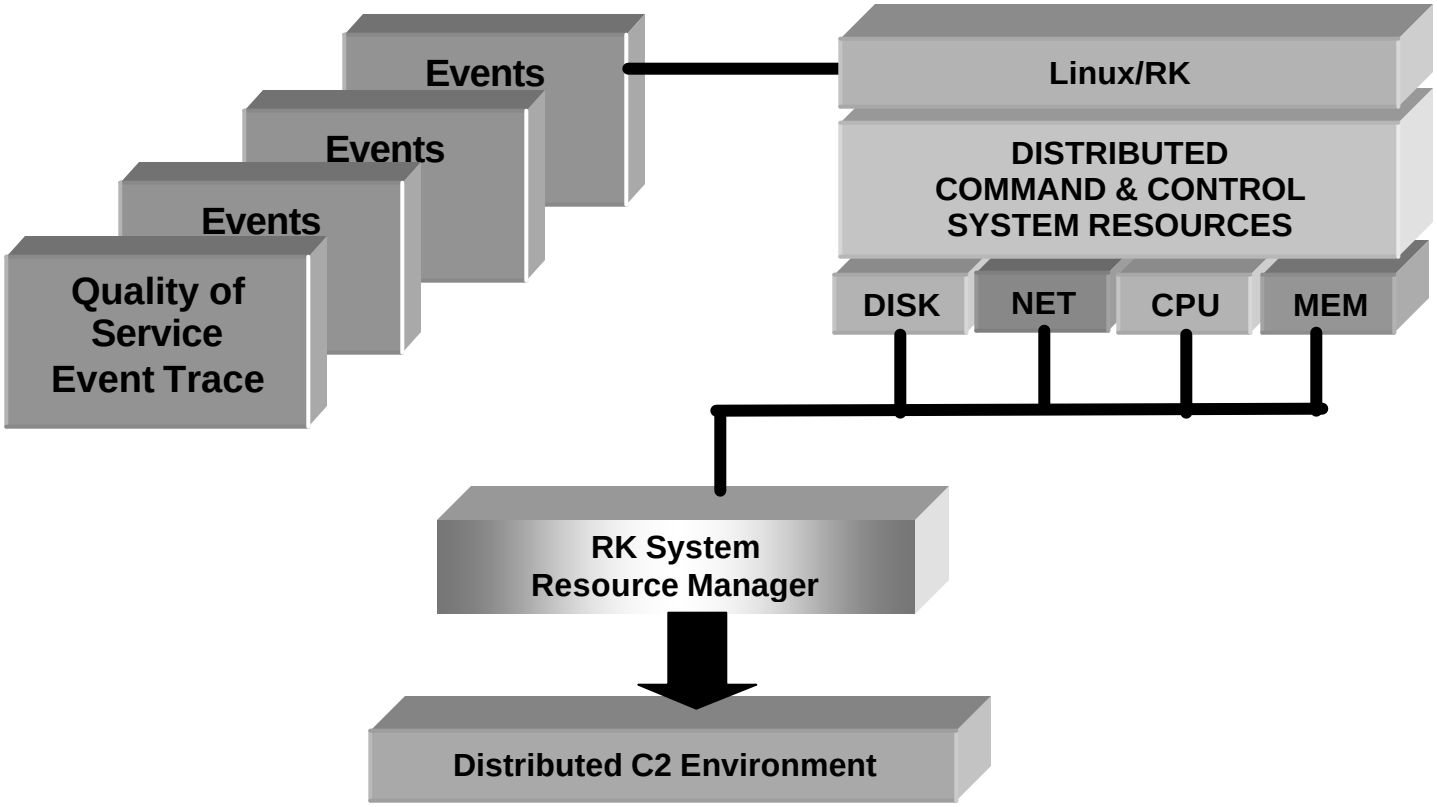

Figure 28. Quality Of Service Resource Management.

The overhead introduced into the targeted failure detection program for the quality of service event trace analysis was minimal despite the fact that an invasive instrumentation method was used.

- FFD Executable alone: 1228288 1228KB
- FFD Executable with ET: 1282437 1282KB
- FFD Object alone: 43748 43.7KB
- FFD Object with ET: 56828 56.8KB

The resulting 54KB approximate size increase however could have some slight influence upon the overall memory utilization and execution time of the target program.

C. PENDING ISSUES

Pending issues for this event trace analysis research include the continued advanced implementation of the Fast Failure Detection system within the NSWC Hiper-D testbed Aegis environment. The inclusion of an expanded number of resource types within the quality of service manager is another pending issue. The current application of the quality of service event trace to the specific case of the targeted failure detection program examined the CPU resource. An effort to increase number of quality of service event trace examinations performed upon the target application program is also a pending issue.

D. FUTURE WORK

Future work in this area of quality of service event trace includes various extensions to the existing analysis program The development dynamic/automated insertion of event trace code callbacks into a perspective targeted system would be an desired feature. The methodology to follow for this extension of the quality of service event trace would entail the development and utilization of three main elements.

- Minimal front-end device to capture the quality of service focus areas (e.g. Event types, etc), and the target specific quality of service calls.
- Development of a tool to allow for the direct instrumentation of event trace callbacks. (or utilization of an existing tool).
- Metric calculation of the event trace analysis results and display of the findings data.

Future work would also include the expansion to the quality of service event trace approach by providing a more formal technique to the development of the quality of service behavioral model. This would be accomplished through the implementation of a specific language from [Auguston 92]. The selection of this language would be drawn from FORMAN, D-Spec, SPEC.

Other future efforts could be directed at expanding the completeness of the behavioral model. The inclusion of actions specific to security into the quality of service event trace framework would add considerably to the behavioral model. This would expand the quality of service metrics into a significantly larger region and scope, however it would follow the quality of service taxonomy described by [Sabata 97] and provide a more complete performance analysis. This behavioral model expansion would require an adjustment during the quality of service event trace phase that isolates the

traceable program points. The identification of these points would include any references that relate to the attainment of required security levels in addition to quality of service levels.

The execution of the quality of service event trace examination upon additional distributed command & control application programs would also be a positive addition to this research. Performing the quality of service event trace examination within multiple quality of service resource management environments (e.g. in addition to the Linux/RK) would add depth to the analysis. Additionally the development of a more user-friendly interface to for inspection of the quality of service event trace output data(e.g. some form of graphic display) would provide an improved analysis evaluation framework.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX

This chapter provides a listing of the data structures and partial source code that has been utilized for the event trace analysis research. The source code documentation has been processed through the Doxygen source documentation application program. This chapter also includes copyright and license information as related to the source code utilized within this dissertation research. Specific sections within this appendix chapter include quality of service event trace data structure documentation, quality of service event trace file documentation, and copyright and license.

A. QOS EVENT TRACE DATA STRUCTURE DOCUMENTATION

css_disk_cpu_reserve_attr Struct Reference

The CPU params for disk access.

`#include <css.h>`

Detailed Description

The CPU params for disk access.

compute_bf; --Timespec structure for CPU needed before the disk access
compute_af; --Timespec structure CPU needed after the disk access
size; --Unsigned long type for file size needed to access
period; --Timespec structure for execution period
deadline; --Timespec structure for execution deadline
blocking_time; --Timespec structure for blocking time required
start_time; --Timespec structure for starting time
reserve_type; –rk_reserve_param_data_t type structure for reservation type

Definition at line 61 of file css.h.

The documentation for this struct was generated from the following file:
css.h

disk_param Struct Reference

The disk parameter needed for CSS decision.

`#include <css.h>`

Detailed Description

The disk parameter needed for CSS decision.

invoke_ticks; --CPU time tick data type for kernel overhead to invoke a request

return_ticks; --CPU time tick data type for kernel overhead to return data back

one_block_ticks; --CPU time tick data type for kernel and I/O overhead for one block access

read_ahead_ticks; --CPU time tick data for kernel overhead to issue one readahead request

user_copy_ticks; --CPU time tick data for user overhead to get one block of data to kernel memory

blocksize; --Unsigned long type indicating block size in Kbytes

max_readahead; --Unsigned long type for indicating maximum disk read ahead in bytes

server_period_ticks; --CPU tick data type for server period

Definition at line 95 of file css.h.

The documentation for this struct was generated from the following file:

css.h

et_qos_error_struct Struct Reference

Event Trace System Data structure for errors and misc.

`#include <event_trace.h>`

Detailed Description

Event Trace System Data structure for errors and misc.

This structure contains the following elements:

int qos_violations -- QoS violations errors

int qos_res_reneg -- QoS specific resource re-negotiation errors

int res_time -- Resource time errors

int cpu_sched -- Cpu scheduling errors

enum et_qos_error_type er_type -- Quality of Service errors during event trace.

Definition at line 240 of file event_trace.h.

The documentation for this struct was generated from the following file:

event_trace.h

et_sys_res Struct Reference

System resource data and information.

`#include <event_trace.h>`

Detailed Description

System resource data and information.

This structure contains the following elements:
float cpu_total; -- Cpu units in compute time
float cpu_used; -- Cpu units utilized
int cpu_avail; -- Cpu units available
int cpu_init; -- Init counting period
int percent_cpu; -- Percentage of CPU used
int disk_total; -- Disk units in size
int disk_used; -- Disk units utilized
int disk_avail; -- Disk units available
int net_total; -- Network units in bandwidth
int net_used; -- Network units utilized
int net_avail; -- Network units available

Definition at line 283 of file event_trace.h.

The documentation for this struct was generated from the following file:
event_trace.h

et_sys_task Struct Reference

Event Trace System Task Data.

`#include <event_trace.h>`

Detailed Description

Event Trace System Task Data.

This structure contains the following elements:

char et_location -- Location of current event trace

char et_task_type -- Current task type eg. main/subroutine/thread

int et_res_resv – Resrouces reserved for this task

cpu_reserve_attr_data_t et_cpu_res_attr --Task res data for CPU resource

Definition at line 174 of file event_trace.h.

The documentation for this struct was generated from the following file:

event_trace.h

et_system_status_struct Struct Reference

Event Trace System Data struct for system reserved resources.

`#include <event_trace.h>`

Detailed Description

Event Trace System Data struct for system reserved resources.

This structure contains the following elements:

char et_rs_name[TRACE_NAME_LEN]; -- Name used to aquire resources

struct **et_sys_task** et_task; -- Task specific data

cpu_reserve_attr_data_t params; -- Task resource data for reservation

Definition at line 198 of file event_trace.h.

The documentation for this struct was generated from the following file:

event_trace.h

event_trace_struct Struct Reference

Main event trace data/statistics.

`#include <event_trace.h>`

Detailed Description

Main event trace data/statistics.

This structure contains the following elements:

struct **et_sys_res** et_res_level; * System resources for reservation *

int et_res_used; * Total system resources utilized *

struct et_task et_event_task; * Task stats for current event trace *

char et_name[TRACE_NAME_LEN]; * Event trace name *

enum et_type et_event_type; * Event type to be recorded *

int et_req_res; * Hist number of resource reservation requests *

int et_res_neg; * Hist number of resource negotiations *

enum et_len et_path_len; * Event trace execution path length (num of funct calls) *

int et_res_req_status; * Resource requests success/failures *

int et_res_type; * Resource type requested (CPU=1 MEM=2 NET=3) *

int et_res_neg_status; * Resource negotiation success/failures *

float et_time_trace; * Message passing timing *

int et_qos_violations; * Violations of QoS *

int et_res_reneg; * Resource re-negotiation times/numbers *

int et_qos_level; * Quality of Service level attained/desired *

int et_error_flag; * List event trace errors *

enum et_loc et_event_loc; * Event locaton as recorded *

int et_sched_policy; * Resource kernal scheduling policy: 0=RM, 1=DM, 2=EDF *

Definition at line 443 of file event_trace.h.

The documentation for this struct was generated from the following file:

event_trace.h

itimerspec Struct Reference

Timer period & timer experation structures.

`#include <posix_timers.h>`

Detailed Description

Timer period & timer experation structures.

Definition at line 19 of file posix_timers.h.

The documentation for this struct was generated from the following file:

posix_timers.h

posix_timer Struct Reference

Posix timer specific structures.

`#include <posix_timers.h>`

Detailed Description

Posix timer specific structures.

Definition at line 59 of file posix_timers.h.

The documentation for this struct was generated from the following file:

posix_timers.h

B. QOS EVENT TRACE FILE DOCUMENTATION

8254.c File Reference

High resolution timer support.

```
#include <linux/config.h>
#include <linux/kernel.h>
#include <asm/io.h>
#include <asm/timex.h>
#include <rk/rk_linux.h>
#include <rk/rk.h>
```

Defines

#define RK_MIN_TIMER 1

Functions

void **rk_update_hw_timer** (struct rk_timer *tmr)
Set the 8254 to go off when the first timer expires.

void **rk_hw_timer_set_periodic** (void)
Set the 8254 to 100Hz (or whatever else HZ is set to) periodic mode.

Detailed Description

High resolution timer support.

8254.c: High resolution timer support for PCs and other machines using the 8254 timer chip (or compatible) in the Linux Resource Kernel.

Definition in file **8254.c**.

Define Documentation

#define RK_MIN_TIMER 1

This defines the minimum # of microseconds used for the timer

Definition at line 36 of file 8254.c.

Referenced by rk_update_hw_timer().

censustaker.c File Reference

Checks for host failure within timeout period.

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <fcntl.h>
#include <limits.h>
#include "ffd.h"
#include "censustaker.h"
#include "hosts.h"
#include "messages.h"
#include "multicast.h"
#include "urgency.h"
#include "event_trace.h"
```

Detailed Description

Checks for host failure within timeout period.

-*- Mode: C -*- **censustaker.c** --- Author : Doug Wells Last Modified By : John Drummond Last Modified On : 2/24/02 Last Machine Used: quorum4 Update Count : 5

censustaker - This program performs the census taker role in the heartbeat function for the QUITE experiment on Fast Failure Detection. It receives periodic messages from other hosts. If a host fails to respond within the timeout period, that host is declared to be dead.

Currently, the program assumes that its local address is the first address returned from DNS for its primary name. If this is not the case, it will be necessary to provide a mechanism by which the proper local interface address can be specified.

The program sends out its state messages to other heartbeat actors via multicast IP. There is currently no mechanism for allowing multiple heartbeat functions to exist simultaneously. The function is limited to the local network, however, by the TTL, which is set to 1. The way to fix this is to provide a mechanism for specifying the multicast address.

Definition in file **censustaker.c**.

censustaker.h File Reference

Include file for checking host failure within timeout period.

Detailed Description

Include file for checking host failure within timeout period.

-*- Mode: C -*- **censustaker.c** --- Author : Doug Wells Last Modified By : Mustafizur Rahman Last Modified On : Thu Aug 16 14:04:33 2001 Last Machine Used: quite01.camb.opengroup.org Update Count : 7

censustaker.h - include file to declare top-level functions for the censustaker part of the Fast Failure Detector. --Doug Wells

Definition in file **censustaker.h**.

cpu_reserve.c File Reference

Resource setup for cpu reservations in the Linux Resource Kernel.

```
#include <rk/rk_linux.h>
#include <rk/rk_error.h>
#include <rk/rk.h>
#include <rk/timespec.h>
#include <linux/time.h>
#include <asm/processor.h>
#include <asm/uaccess.h>
#include <asm/signal.h>
```

Defines

#define **CPU_CAPACITY_MAX** PERCENT2CAPACITY(66)

Functions

void **rk_resource_set_schedule** (struct rs_proc_list *rs_proc, unsigned int arg)

Reschedule the eligible resource set with highest priority (top of the rs_list). If no eligible resource set is available, halt the task. Otherwise, update the scheduling parameter of the task.

void **rk_resource_set_adjust_accounting** (void)

Update the accounting of the current task if its scheduling parameter changes.

int **cpu_reserve_eligible** (rk_reserve_t rsv)

Check the eligibility of the cpu reserve.

int **cpu_reserve_destroy** (rk_reserve_t)

Destroy CPU reserves that have been previously created.

void **cpu_reserve_start_account** (rk_reserve_t)

Start & Stop the account of CPU utilization.

void **cpu_reserve_stop_account** (rk_reserve_t, cpu_tick_t)

Stop the accounting for the CPU reserves that have been allocated.

void **cpu_reserve_replenish** (rk_reserve_t, cpu_tick_t, cpu_tick_t)

Replenish & Enforce a reserve. Add parameter start_ticks which is the start time of each period of the reserve. This is for computing the eligible deadline of the reserve.

void **cpu_reserve_enforce** (rk_reserve_t)

Enforce the CPU reserves that have been allocated.

void **cpu_reserve_sleep_on** (rk_reserve_t)

Apply sleep to CPU reserves list elements.

void **cpu_reserve_attach** (rk_reserve_t, struct rs_proc_list *)

Attach CPU reserve process.

void **cpu_reserve_detach** (rk_reserve_t, struct rs_proc_list *)
Detach CPU reserve process.

void **cpu_reserve_update_ticket** (rk_reserve_t, unsigned long)
Update the CPU reserve cpu ticks.

void **cpu_reserve_ticket_query** (rk_reserve_t, unsigned long *)
Get the CPU reserves tick info.

void **cpu_reserve_wait_one_ticket** (rk_reserve_t)
Wait time for CPU tick.

int **admit_reserve_request** (cpu_reserve_t cpu)
Admission control for CPU reserves.

void **priority_list_add** (cpu_reserve_t cpu, struct list_head *head)
Add specified reservation into reservation list in priority order. Adjust reservation_priority index of list members accordingly.

int **priority_list_remove** (cpu_reserve_t cpu, struct list_head *head)
Remove specified reservation from reservation list. Adjust reservation_priority index of remaining list members accordingly.

int **efficient_timespec_ceil** (struct timespec dividend, struct timespec divider)
This ceiling computation works faster by assuming that the seconds field is less than 1000 seconds, and by truncating the nanoseconds field. If the seconds field is greater than 1000 seconds, a valid but VERY inefficient response will result!

void **cpu_reserve_init** (void)
Initialize the CPU reservation list.

rk_reserve_t **cpu_reserve_create** (rk_resource_set_t rs, cpu_reserve_attr_t cpu_attr)
Create CPU reserves.

int **cpu_reserve_ctl** (rk_reserve_t rsv, cpu_reserve_attr_t cpu_attr)
CPU reserves chagnes.

void **cpu_reserve_summation_of_used_ticks** (cpu_reserve_t cpu, cpu_tick_t now)
Add used CPU ticks that have been reserved.

void **cpu_reserve_sched_enable** (struct rs_proc_list *rs_proc, unsigned int arg)
Enable scheduler to make a process eligible for execution.

void **cpu_reserve_sched_disable** (struct rs_proc_list *rs_proc, unsigned int arg)
Disable scheduler from making process eligible for execution.

void **cpu_reserve_check_enforce** (rk_reserve_t rsv, cpu_reserve_t cpu, cpu_tick_t now)
Check the CPU reserve enforcement.

int **cpu_reserve_delete** (rk_resource_set_t rs)
Delete CPU reserves that have been previously setup.

rk_reserve_t **cpu_reserve_dummy_create** (rk_resource_set_t rs, time_t duration)
The cpu reserve for measurement only. This is for default and idle resource sets. No need to apply admission control to these reserves. We apply the 100% capacity reserve. The parameter duration is the measurement granularity we want to keep track the cpu usage.

Detailed Description

Resource setup for cpu reservations in the Linux Resource Kernel.

cpu_reserve.c: code to support CPU reservations

Pittsburgh PA 15213-3890

or via email to raj@ece.cmu.edu

any improvements or extensions that they make and grant Carnegie Mellon the rights to redistribute these changes.

Definition in file **cpu_reserve.c**.

Define Documentation

#define CPU_CAPACITY_MAX PERCENT2CAPACITY(66)

66% ... is the maximum available capacity for reserves.

Definition at line 146 of file cpu_reserve.c.

Variable Documentation

struct rk_reserve_ops cpu_reserve_ops

Initial value:

```
 {
    cpu_reserve_destroy,
    cpu_reserve_start_account,
    cpu_reserve_stop_account,
    cpu_reserve_replenish,
    cpu_reserve_enforce,
    cpu_reserve_attach,
    cpu_reserve_detach,




    cpu_reserve_update_ticket,
    cpu_reserve_ticket_query,
    cpu_reserve_wait_one_ticket,
}
```

Definition at line 120 of file cpu_reserve.c.

css.h File Reference

Sets up includes for cpu & disk access rsources in the Linux Resource Kernel.

`#include <rk/rk.h>`

Data Structures

struct css_disk_cpu_reserve_attr

The CPU params for disk access.

struct disk_param

The disk parameter needed for CSS decision.

Functions

int **css_disk_cpu_reserve_create** (rk_resource_set_t, css_disk_cpu_reserve_attr_t)

Create the reservation for access a disk w/ cpu needed by the application before and after accessing the disk.

Detailed Description

Sets up includes for cpu & disk access rsources in the Linux Resource Kernel.

CSS for cpu & disk access. Assume that the application does some computation then access the disk and does some computation after the disk access. This task model can be changed later.

Definition in file **css.h**.

css_disk_cpu.c File Reference

Disk utilization of CPU resource for the Linux Resource Kernel.

```
#include <rk/rk_linux.h>
#include <rk/rk_error.h>
#include <rk/rk.h>
#include <rk/timespec.h>
#include <linux/time.h>
#include <asm/processor.h>
#include <asm/uaccess.h>
#include <rk/css.h>
```

Defines

#define **X_SLACK** 50

Functions

void **tick2timespec** (long, struct timespec *)
Timekeeping function.

int **css_disk_cpu_reserve_create** (rk_resource_set_t rs, css_disk_cpu_reserve_attr_t css_attr)
Create the reservation for access a disk w/ cpu needed by the application before and after accessing the disk.

asmlinkage int **sys_css_disk_cpu_reserve_create** (rk_resource_set_t rs, css_disk_cpu_reserve_attr_t css_attr)
System call for disk_cpu reservation create.

Detailed Description

Disk utilization of CPU resource for the Linux Resource Kernel.

Pittsburgh PA 15213-3890

or via email to raj@ece.cmu.edu

any improvements or extensions that they make and grant Carnegie Mellon the rights to redistribute these changes.

Definition in file **css_disk_cpu.c**.

Define Documentation

#define X_SLACK 50

Proportion of disk slack over the summation of bf+af slack (in percents)

Definition at line 43 of file css_disk_cpu.c.

Referenced by css_disk_cpu_reserve_create().

disk_reserve.c File Reference

Resource setup for disk reservations in the Linux Resource Kernel.

```
#include <rk/rk_linux.h>
#include <rk/rk_error.h>
#include <rk/rk.h>
#include <rk/timespec.h>
#include <linux/time.h>
#include <asm/uaccess.h>
#include <rk/css.h>
#include <linux/fs.h>
#include <linux/blkdev.h>
```

Defines

#define **CSS_CONFIG** 1

Functions

void **disk_reserve_replenish** (rk_reserve_t, cpu_tick_t, cpu_tick_t)
Replenish & Enforce a reserve.

void **disk_reserve_update_account** (rk_reserve_t, unsigned long)
Update the used blocks; need to convert size in KBytes to blocks.

void **disk_reserve_quota_query** (rk_reserve_t, unsigned long *)
Ask for the available quota.

int **disk_reserve_admit_request** (disk_reserve_attr_t)
Admission control for DISK reserves.

int **disk_reserve_adm_cancel_request** (disk_reserve_t disk)
Update the state of the admission due to request cancellation.

void **disk_size_to_blocks** (unsigned long, disk_block_t *)
Convert size in KB to number of blocks.

int **disk_capacity** (unsigned long, cpu_tick_data_t)
Compute the disk capcacity of the request.

void **rk_disk_reserve_attach_bh** (struct task_struct *, struct buffer_head *)
If there is a resource set already attached to the buffer head, use the resource set that has disk reserve w/ higher priority. In case that there is no resource set attached to the buffer head, attach the resource set that has disk reserve. Otherwise attach the tsk current resource set to the buffer head.

int **rk_disk_readahead_max** (struct inode *, struct task_struct *)
Define the readahead blocks In Linux kernel, fs tries to readahead for whole size of request or max_readahead specified by device (default:MAX_READAHEAD defined in blkdev.h).

int **disk_higher_prio** (rk_reserve_t, rk_reserve_t)
Check if it's higher priority.

rk_reserve_t **rk_get_disk_reserve** (struct task_struct *)
Use disk_reserve that attaches to the current resource set of the task. Otherwise, use the disk reserve of the highest priority resource set.

void **disk_reserve_init** (void)
Initialize DISK reserves.

int **disk_server_start** (struct task_struct *tsk)
Start disk resource sever.

void **disk_server_wait_for_ticket** ()
Get one ticket to run.

int **disk_server_replenish_register** (void(*f)(rk_reserve_t, unsigned long))
Register replenish function.

int **disk_server_replenish_unregister** (void)
Unregister replenish function.

rk_reserve_t **disk_reserve_create** (rk_resource_set_t rs, disk_reserve_attr_t disk_attr)
Create the DISK reserve.

int **max_disk_blocks** (struct timespec time)
Compute number of bocks can read in the duration w/ full capacity.

linux rk_reserve_t **disk_reserve_dummy_create** (rk_resource_set_t rs, time_t duration)
The disk reserve for measurement only. This is for default resoruce set. No need to apply admission control to these reserves. We apply the 100% capacity reserve. The parameter duration is the measurement granularity we want to keep track the cpu usage.

Detailed Description

Resource setup for disk reservations in the Linux Resource Kernel.

disk_reserve.c: code to support DISK reservations

WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this software; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

This file is derived from software distributed under the following terms:

Real-time and Multimedia Systems Laboratory Copyright (c) 2000 Carnegie Mellon University All Rights Reserved.

Permission to use, copy, modify and distribute this software and its documentation is hereby granted, provided that both the copyright notice and this permission notice appear in all copies of the software, derivative works or modified versions, and any portions thereof, and that both notices appear in supporting documentation.

CARNEGIE MELLON ALLOWS FREE USE OF THIS SOFTWARE IN ITS "AS IS" CONDITION. CARNEGIE MELLON DISCLAIMS ANY LIABILITY OF ANY KIND FOR ANY DAMAGES WHATSOEVER RESULTING FROM THE USE OF THIS SOFTWARE.

Carnegie Mellon requests users of this software to return to

Real-Time and Multimedia Systems Laboratory Attn: Prof. Raj Rajkumar Electrical and Computer Engineering, and Computer Science Carnegie Mellon University Pittsburgh PA 15213-3890

or via email to `raj@ece.cmu.edu`

any improvements or extensions that they make and grant Carnegie Mellon the rights to redistribute these changes.

Definition in file **disk_reserve.c**.

Define Documentation

#define CSS_CONFIG 1

Unable CSS Functinality for this moment

Definition at line 99 of file disk_reserve.c.

Variable Documentation

struct rk_reserve_ops disk_reserve_ops

Initial value:

```
{
        disk_reserve_destroy,
        NULL,
        NULL,
        disk_reserve_replenish,
        disk_reserve_enforce,
        NULL,
        NULL,
```

```
        disk_reserve_update_account,
        disk_reserve_quota_query,
        NULL,
}
```

Definition at line 77 of file disk_reserve.c.

division.c File Reference

Support file for 64-bit division in the Linux Resource Kernel.

```
#include "longlong.h"
```

Data Structures

struct DIstruct
union DIunion

Detailed Description

Support file for 64-bit division in the Linux Resource Kernel.

This file contains excerpts of libgcc to support 64-bit division without having to link against libgcc. NOTE: More subroutines needed by GCC output code on some machines. Compile this one with gcc. Copyright (C) 1989, 92-98, 1999 Free Software Foundation, Inc.

This file is part of GNU CC.

GNU CC is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2, or (at your option) any later version.

GNU CC is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with GNU CC; see the file COPYING. If not, write to the Free Software Foundation, 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA. *

As a special exception, if you link this software with other files, some of which are compiled with GCC, to produce an executable, this software does not by itself cause the resulting executable to be covered by the GNU General Public License. This exception does not however invalidate any other reasons why the executable file might be covered by the GNU General Public License. *

It is incorrect to include config.h here, because this file is being compiled for the target, and hence definitions concerning only the host do not apply.

Definition in file **division.c**.

Variable Documentation

const UQItype __clz_tab[] `[static]`

Initial value:

```
{
   0, 1, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5,
   5, 5, 5, 5, 5, 5, 5, 5,
   6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
   6, 6, 6, 6, 6, 6, 6, 6,
   7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7,
```

```
    7, 7, 7, 7, 7, 7, 7, 7,
    7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7,
    7, 7, 7, 7, 7, 7, 7, 7,
    8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
    8, 8, 8, 8, 8, 8, 8, 8,
    8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
    8, 8, 8, 8, 8, 8, 8, 8,
    8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
    8, 8, 8, 8, 8, 8, 8, 8,
    8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
    8, 8, 8, 8, 8, 8, 8, 8,
}
```

Definition at line 160 of file division.c.

event_trace.c File Reference

Initial startup file for the QoS event trace.

```
#include "event_trace.h"
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include "utils.h"
```

Functions

int **et_system_data** (int et_rk_sys)

Retrieve all the resource kernel system reserve data, which includes the system and other resource sets. This function returns an int upon completion: 0=success, -1=unsuccessful.

int **et_dump_data** (int i)

Output event trace statistics to I/O device This function returns an int upon completion: 0=success, -1=unsuccessful.

int et_struct_reset (struct event_trace_struct *et_data_status)

Reset elements of the global data structure, and retain the dynamic totals within this structure. This function returns an int upon completion: 0=success, -1=unsuccessful.

float **et_time_calc** (struct timespec *et_time)

Calculate micro second data based upon the received timespec structure. This function returns an unsigned int which represents the ms time value upon successful completion, if unsuccessful returns 0.

int event_trace_tab (struct event_trace_struct *et_data_tab)

Tabulate the system quality of service data here, and keeps dynamic totals within the et_data data structure. This function returns an int upon completion: 0=success, -1=unsuccessful.

int event_trace_init (struct event_trace_struct *et_init)

Initialize all event trace params here, and allocate memory for structure This function returns an int 0 if successful, -1 if unsuccessful.

int event_trace_record (struct event_trace_struct *et_data_status)

Record QoS trace event, update data information within event trace structure. Returns updated error condition.

Detailed Description

Initial startup file for the QoS event trace.

File : **event_trace.c** Author : John Drummond Last Modified : 2/24/02

Notes : This file is written in standard C language, the purpose of which is to provide a traceing for various Quality of Service Events (QoSE) execution path calls.

Definition in file **event_trace.c**.

Function Documentation

int et_struct_reset (struct event_trace_struct * *et_data_status*)

Reset elements of the global data structure, and retain the dynamic totals within this structure. This function returns an int upon completion: 0=success, -1=unsuccessful.

/*!

Definition at line 284 of file event_trace.c.

```
285 {
286
287         int reset_ret = 0;
288
289         #ifdef EVENT_TRACE_DEBUG3
290         printf("Reset Globals\n");
291         #endif
292
293         et_ffd->et_event_task.et_cpu_res_attr.compute_time.tv_sec   = 0x00;
294         et_ffd->et_event_task.et_cpu_res_attr.compute_time.tv_nsec  = 0x00;
295
296         et_ffd->et_event_task.et_cpu_res_attr.period.tv_sec         = 0x00;
297         et_ffd->et_event_task.et_cpu_res_attr.period.tv_nsec        = 0x00;
298
299         et_ffd->et_event_task.et_cpu_res_attr.deadline.tv_sec       = 0x00;
300         et_ffd->et_event_task.et_cpu_res_attr.deadline.tv_nsec      = 0x00;
301
302         et_ffd->et_event_task.et_cpu_res_attr.reserve_type.enf_mode = 0x00;
303         et_ffd->et_event_task.et_cpu_res_attr.reserve_type.rep_mode = 0x00;
304         et_ffd->et_event_task.et_cpu_res_attr.reserve_type.sch_mode = 0x00;
305
306         et_ffd->et_res_level.cpu_used       = 0x00;
307 //et_ffd->et_res_level.cpu_avail      = 0x00;
308 //et_ffd->et_res_level.cpu_init       = 0x00;
309         et_ffd->et_res_level.percent_cpu    = 0x00;
310
311         et_ffd->et_res_level.disk_used      = 0x00;
312         et_ffd->et_res_level.disk_avail     = 0x00;
313
314         et_ffd->et_res_level.net_used       = 0x00;
315         et_ffd->et_res_level.net_avail      = 0x00;
316
317         return reset_ret;
318
319 }; /* ET_STRUCT_RESET */
```

int et_system_data (int *et_sys*)

Retrieve all the resource kernel system reserve data, which includes the system and other resource sets. This function returns an int upon completion: 0=success, -1=unsuccessful.

/*!

Definition at line 132 of file event_trace.c.

References et_system_status_struct::params.

```
133 {
134     int i, k, ncount, count1 = 0, count2 = 0, rcount =0;
135     char name[64];
136     rk_resource_set_t *rsets;
137     rk_reserve_t rsv;
138     cpu_reserve_attr_data_t params;
139     pid_t *procs;
140 
141         while (1) {
142                 count1 = rk_resource_sets_get_num();
143 
144                 #ifdef EVENT_TRACE_DEBUG3
145                 printf("system has %d resource sets\n", count1);
146                 #endif
147 
148                                                         /* Allocate mem for system
status structure */
149                 if((et_sys=(struct et_system_status_struct*) malloc(count1*
sizeof(struct et_system_status_struct)))===NULL){
150                         fprintf (stderr, "EVENT_TRACE ERROR:FFD.C_MAIN et_sys
malloc\n");
151                         return -1;
152                         //++errflag;
153                 }
154                 bzero(et_sys, sizeof(struct et_system_status_struct));
155                 rsets = (rk_resource_set_t *) malloc(count1 *
sizeof(rk_resource_set_t));
156                  if ( (count2 = rk_resource_sets_get_list(rsets, count1 *
sizeof(rk_resource_set_t))) == count1) {
157                         /* count not changing anymore! */
158                         break;
159                 }
160                 else {
161                         if (count2 < 0) {
162                                 fprintf(stderr, " could not get rset information!
(error = %d)\n", count2);
163                                 exit(-2);
164                         }
165                         fprintf(stderr, "new return value: %d, original count:
%d\n", count2, count1);
166                 } /*ELSE*/
167 
168                 free(rsets);
169 
170         } /*WHILE*/
171 
172                                                                                 /*
Get the details of each rset */
173         for (i = 0; i < count1; i++) {
174 
175                 rk_resource_set_get_name(rsets[i], name);
176 
177                                                 /* Skip the rset redundant rsets */
178 
if(((ncount=strcmp(name,"default"))==0)||((ncount=strcmp(name,"idle"))==0)||((ncount=s
trcmp(name,"ffd"))==0)) {
179                         ; /* SKIP NON-ESSENTIAL RESOURCE SETS */
180                         #ifdef EVENT_TRACE_DEBUG1
181                         printf(">>>>>>>>>>>>>> SKIPPING Resource set name= %s \n",
name);
182                         #endif
183                 }
184                 else {
185                         strcpy(et_sys[rcount].et_rs_name, name);                /*
Get reserve name */
186 
187                         #ifdef EVENT_TRACE_DEBUG1
188                         printf("ASSIGNING Resource set name= %s rcount= %d i=
%d\n", name, rcount, i);
```

```
189                         #endif
190                         rsv = rk_resource_set_get_cpu_rsv(rsets[i]);           /*
Get CPU reserves */
191                         if (rsv != NULL) {
192
193                         #ifdef EVENT_TRACE_DEBUG1
194                         printf("rsv!=NULL rk_resource_set_get_cpu_rsv Resource set
name=%s rcount=%d i=%d\n",name,rcount,i);
195                         #endif
196                                 if (rk_cpu_reserve_get_attr(rsv, &params) !=
RK_SUCCESS) {
197                                         #ifdef EVENT_TRACE_DEBUG1
198                                         printf("RK_FAILURE get_attr Res set
name=%s rcount=%d i=%d\n",name,rcount,i);
199                                         #endif
200                                         fprintf(stderr, " failed to get attributes
of CPU reserve %p\n", rsv);
201                                 }
202                                 else { /*RK_SUCCESS*/
203
204                                 #ifdef EVENT_TRACE_DEBUG1
205                                 printf("RK_SUCCESS  Resource set name= %s rcount=
%d i= %d\n", name, rcount, i);
206                                 #endif
207                                                                                 /*
COPY CPU RESERVE ATTRIBUTE DATA */
208                                 et_sys[rcount].params.compute_time.tv_nsec   =
params.compute_time.tv_nsec;
209                                 et_sys[rcount].params.compute_time.tv_nsec   =
params.compute_time.tv_nsec;
210                                 et_sys[rcount].params.period.tv_sec          =
params.period.tv_sec;
211                                 et_sys[rcount].params.period.tv_nsec         =
params.period.tv_nsec;
212                                 et_sys[rcount].params.deadline.tv_sec        =
params.deadline.tv_sec;
213                                 et_sys[rcount].params.deadline.tv_nsec       =
params.deadline.tv_nsec;
214                                 et_sys[rcount].params.reserve_type.enf_mode  =
params.reserve_type.enf_mode;
215                                 et_sys[rcount].params.reserve_type.rep_mode  =
params.reserve_type.rep_mode;
216                                 et_sys[rcount].params.reserve_type.sch_mode  =
params.reserve_type.sch_mode;
217
218                                 ++rcount;
219
220                                 #ifdef EVENT_TRACE_DEBUG1
221                                 printf("ENDING LOOP Resource set name= %s rcount=
%d i= %d\n", name, rcount, i);
222                                 #endif
223                                 } /*ELSE RK_SUCCESS*/
224                         } /*IF RSV*/
225                         else
226                         printf("rsv==NULL rk_resource_set_get_cpu_rsv Resource set
name=%s rcount=%d i=%d\n",name,rcount,i);
227                 } /*ELSE idle/default */
228
229         } /*FOR COUNT1*/
230
231         return rcount;
232
233 }; /* ET_SYSTEM_DATA */
```

float et_time_calc (struct timespec * *et_time*)

Calculate micro second data based upon the received timespec structure. This function returns an unsigned int which represents the ms time value upon successful completion, if unsuccessful returns 0.

/*!

Parameters:

struct timespec et_time.

Returns:

Float, if successful timespec structure, if unsuccessful 0.

Definition at line 329 of file event_trace.c.

```
330 {
331
332         float et_time_us;
333
334         #ifdef EVENT_TRACE_DEBUG3
335         printf("Calculating Timespec\n");
336         #endif
337
338
339         if (et_time == NULL)
340                 return (0.0);
341
342
343         et_time_us = et_time->tv_sec * 1000000 + et_time->tv_nsec / 1000;
344
345         return (et_time_us/1000);
346
347 }; /* ET_TIME_CALC */
```

int event_trace_init (struct event_trace_struct * *et_init*)

Initialize all event trace params here, and allocate memory for structure This function returns an int 0 if successful, -1 if unsuccessful.

/*!

Parameters:

Char for name of event trace.

Returns:

An int type if successful 0, if unsuccessful -1

Definition at line 405 of file event_trace.c.

```
406 {
407
408         int init_ret = 0;
409
410         #ifdef EVENT_TRACE_DEBUG3
411         printf("Initializing Event Trace Params\n");
412         #endif
413
414                                 /* Allocate mem for the event trace structure */
415         if ((et_init = (struct event_trace_struct *) malloc(sizeof(struct
event_trace_struct))) == NULL) {
416            fprintf (stderr, "ET ERROR:EVENT_TRACE.C_EVENT_TRACE_INIT malloc()\n");
417            return -1;
418         }
419
420
421                                                         /*Zero reserve attribute
data */
422         et_init->et_event_task.et_cpu_res_attr.compute_time.tv_sec  = 0x00;
```

```
423         et_init->et_event_task.et_cpu_res_attr.compute_time.tv_nsec = 0x00;
424         et_init->et_event_task.et_cpu_res_attr.period.tv_sec        = 0x00;
425         et_init->et_event_task.et_cpu_res_attr.period.tv_nsec       = 0x00;
426         et_init->et_event_task.et_cpu_res_attr.deadline.tv_sec      = 0x00;
427         et_init->et_event_task.et_cpu_res_attr.deadline.tv_nsec     = 0x00;
428
429         et_init->et_event_task.et_cpu_res_attr.reserve_type.enf_mode = 0x00;
430         et_init->et_event_task.et_cpu_res_attr.reserve_type.rep_mode = 0x00;
431         et_init->et_event_task.et_cpu_res_attr.reserve_type.sch_mode = 0x00;
432
433                                                      /* Zero resource totals */
434         et_init->et_res_level.cpu_total      = 0x00;
435         et_init->et_res_level.cpu_used       = 0x00;
436         et_init->et_res_level.cpu_avail      = 0x00;
437         et_init->et_res_level.disk_total     = 0x00;
438         et_init->et_res_level.disk_used      = 0x00;
439         et_init->et_res_level.disk_avail     = 0x00;
440         et_init->et_res_level.net_total      = 0x00;
441         et_init->et_res_level.net_used       = 0x00;
442         et_init->et_res_level.net_avail      = 0x00;
443
444         et_init->et_res_req_status           = 0x00;   /* Zero request
success/failure */
445         et_init->et_event_loc                = 0x00;   /* Zero event location */
446         et_init->et_time_trace               = 0x00;   /* Zero internal trace time
clock record*/
447         et_init->et_sched_policy             = 0x00;   /* Resource kernal
scheduling policy: 0=RM, 1=DM, 2=EDF */
448         et_init->et_event_task.et_trace_lev  = 0x00;   /* Zero trace level */
449
450 //strcpy(et_data->et_event_task.et_location, et_data_status-
>et_event_task.et_location); /* Update event location*/
451 //strcpy(et_data->et_event_task.et_task_type, et_data_status-
>et_event_task.et_task_type);
452
453                                                      /* Zero out the struct
members */
454         et_init->et_event_type      = ET_NULL;         /* Event type to be
recorded */
455         et_init->et_req_res         = 0x00;            /* Hist number of resource
reservation requests */
456         et_init->et_res_neg         = 0x00;            /* Hist number of resource
negotiations */
457         et_init->et_path_len        = 0x00;            /* Event trace execution
path length (num of funct calls) */
458         et_init->et_res_req_status  = 0x00;            /* Resource requests
success/failure */
459         et_init->et_res_type        = 0x00;            /* Resource type requested
(CPU=1 MEM=2 NET=3) */
460         et_init->et_res_neg_status  = 0x00;            /* Resource negotiation
success/failures */
461
462         et_init->et_time_trace      = 0x00;            /* Message passing timing
*/
463         et_init->et_qos_violations  = 0x00;            /* Violations of QoS */
464         et_init->et_res_reneg       = 0x00;            /* Resource re-negotiation
times/numbers */
465         et_init->et_qos_level       = 0x00;            /* Quality of Service level
attained/desired */
466
467         return init_ret;
468
469 }; /* EVENT_TRACE_INIT */
```

int event_trace_record (struct event_trace_struct * *et_data_status*)

Record QoS trace event, update data information within event trace structure. Returns updated error condition.

/*!

Parameters:

Structure event_trace_t for recording event data.

Returns:

Error condition integer indicating success/failure.

Definition at line 477 of file event_trace.c.

```
478 {
479         struct event_trace_struct *et_data;
480
481         char    buff [TRACE_TYPE_LEN];
482         double  curtime;
483         struct timeval  tv;
484
485         int et_write = 0;
486         int et_send = 0;
487         int i = 0;
488         int et_tab_sd =0;
489
490         int sys_ret =0;
491
492         #ifdef EVENT_TRACE_DEBUG3
493         printf("Recording Events\n");
494         #endif
495
496                                                         /* Allocate mem for the
event trace structure */
497 /*JD From arch/i387/kernel/irq.c -action = (struct irqaction
*)kmalloc(sizeof(struct irqaction),GFP_KERNEL);*/
498 //if ((et_data = (struct event_trace_struct *) kmalloc(sizeof(struct
event_trace_struct),NULL )) == NULL)
499         if (!(et_data = (struct event_trace_struct *) malloc(sizeof(struct
event_trace_struct))))
500                 return -2;
501
502         bzero(et_data, sizeof(struct event_trace_struct));
503
504                                                         /*Copy reserve attribute
data */
505 et_data->et_event_task.et_cpu_res_attr.compute_time.tv_sec = et_data_status-
>et_event_task.et_cpu_res_attr.compute_time.tv_sec;
506 et_data->et_event_task.et_cpu_res_attr.compute_time.tv_nsec = et_data_status-
>et_event_task.et_cpu_res_attr.compute_time.tv_nsec;
507 et_data->et_event_task.et_cpu_res_attr.period.tv_sec = et_data_status-
>et_event_task.et_cpu_res_attr.period.tv_sec;
508 et_data->et_event_task.et_cpu_res_attr.period.tv_nsec = et_data_status-
>et_event_task.et_cpu_res_attr.period.tv_nsec;
509 et_data->et_event_task.et_cpu_res_attr.deadline.tv_sec = et_data_status-
>et_event_task.et_cpu_res_attr.deadline.tv_sec;
510 et_data->et_event_task.et_cpu_res_attr.deadline.tv_nsec = et_data_status-
>et_event_task.et_cpu_res_attr.deadline.tv_nsec;
511
512 et_data->et_event_task.et_cpu_res_attr.reserve_type.enf_mode = et_data_status-
>et_event_task.et_cpu_res_attr.reserve_type.enf_mode;
513 et_data->et_event_task.et_cpu_res_attr.reserve_type.rep_mode = et_data_status-
>et_event_task.et_cpu_res_attr.reserve_type.rep_mode;
514 et_data->et_event_task.et_cpu_res_attr.reserve_type.sch_mode = et_data_status-
>et_event_task.et_cpu_res_attr.reserve_type.sch_mode;
515                                                         /* Copy resource totals */
516         et_data->et_res_level.cpu_total      =  et_data_status-
>et_res_level.cpu_total;
517         et_data->et_res_level.cpu_used       =  et_data_status-
>et_res_level.cpu_used;
518 //et_data->et_res_level.cpu_avail      =        et_data_status-
>et_res_level.cpu_avail;
```

```
519         et_data->et_res_level.cpu_init      =   et_data_status-
>et_res_level.cpu_init;
520
521 //printf("BEFORE SYSTEM RES:et_data_statusCPU_LEVEL=%d
et_data_statusCOMP_TIME=%3.3f et_dataCPU_LEVEL=%d
et_dataCOMP_TIME=%3.3f\n",et_data_status->et_res_level.cpu_avail,
et_time_calc(&(et_data_status->et_event_task.et_cpu_res_attr.compute_time)), et_data-
>et_res_level.cpu_avail, et_time_calc(&(et_data-
>et_event_task.et_cpu_res_attr.compute_time)));
522
523 //sys_ret =  et_get_sys_res(et_data);
524 sys_ret =  et_get_sys_res(et_data_status);
525
526 et_data->et_res_level.cpu_avail = et_data_status->et_res_level.cpu_avail;
527 et_data->et_res_level.percent_cpu = et_data_status->et_res_level.percent_cpu;
528
529 //printf("AFTER  SYSTEM RES:et_data_statusCPU_LEVEL=%d
et_data_statusCOMP_TIME=%3.3f et_dataCPU_LEVEL=%d
et_dataCOMP_TIME=%3.3f\n",et_data_status->et_res_level.cpu_avail,
et_time_calc(&(et_data_status->et_event_task.et_cpu_res_attr.compute_time)), et_data-
>et_res_level.cpu_avail, et_time_calc(&(et_data-
>et_event_task.et_cpu_res_attr.compute_time)));
530
531         et_data->et_res_level.disk_total     =  et_data_status-
>et_res_level.disk_total;
532         et_data->et_res_level.disk_used      =  et_data_status-
>et_res_level.disk_used;
533         et_data->et_res_level.disk_avail     =  et_data_status-
>et_res_level.disk_avail;
534         et_data->et_res_level.net_total      =  et_data_status-
>et_res_level.net_total;
535         et_data->et_res_level.net_used       =  et_data_status-
>et_res_level.net_used;
536         et_data->et_res_level.net_avail      =  et_data_status-
>et_res_level.net_avail;
537
538         et_data->et_res_req_status          = et_data_status->et_res_req_status;
/* Update request success/failure */
539         et_data->et_event_loc               = et_data_status->et_event_loc; /*
Update event level */
540         et_data->et_event_task.et_trace_lev  = et_data_status-
>et_event_task.et_trace_lev; /* Update event level */
541         et_data->et_event_type              = et_data_status->et_event_type; /*
Update event type */
542         et_data->et_req_res                 = et_data_status->et_req_res; /*
Update number of res reservation requests */
543         et_data->et_res_neg                 = et_data_status->et_res_neg; /*
Update number of resource negotiations */
544         et_data->et_path_len                = +et_data_status->et_path_len;  /*
Add to trace execution path length  */
545         et_data->et_qos_level               = et_data_status->et_qos_level; /*
Quality of Service level attained/desired */
546         et_data->et_time_trace              = et_data_status->et_time_trace; /*
Updata internal trace time clock record*/
547         et_data->et_qos_violations          = et_data_status->et_qos_violations;
/* Record violation of QoS */
548         et_data->et_res_reneg               = et_data_status->et_res_reneg;
/*Update resource re-negotiation data */
549         strcpy(et_data->et_event_task.et_location, et_data_status-
>et_event_task.et_location); /* Update event location*/
550         strcpy(et_data->et_event_task.et_task_type, et_data_status-
>et_event_task.et_task_type);
551
552         if (gettimeofday (&tv, NULL) == -1)
553                 perror ("gettimeofday");
554                 curtime = cv_timeval_to_double (tv);
555                 et_data->et_time_trace = curtime;                       /* Message
passing timing */
556                 sprintf(JDfilename, "JD-ET%d", 100);
557                 fout = fopen(JDfilename, "a");
558                 //fprintf(fout, "\nCURRENT EVENT TRACE TIME: %.3f:\n",curtime);
```

```
559
560         switch (et_data_status->et_event_type)
561         {
562         case SYS_RES:
563                 strcpy(buff, "SYS_RES");
564                                                                         /* Get
only system statistics */
565                 if ((et_send = et_system_data(et_send)) != -1) {
566 //printf("\n>>>>>>>>>>>>>>>>>>>>SWITCH EVENT_TYPE et_send= %d\n", et_send);
567                         for (i = 0; i < et_send; i++) {
568                                                                         /*Copy
only system stats */
569                         et_data->et_event_task.et_cpu_res_attr.compute_time.tv_sec
= et_sys[i].params.compute_time.tv_sec;
570                         et_data-
>et_event_task.et_cpu_res_attr.compute_time.tv_nsec =
et_sys[i].params.compute_time.tv_nsec;
571
572                         et_data->et_event_task.et_cpu_res_attr.period.tv_sec =
et_sys[i].params.period.tv_sec;
573                         et_data->et_event_task.et_cpu_res_attr.period.tv_nsec =
et_sys[i].params.period.tv_nsec;
574
575                         et_data->et_event_task.et_cpu_res_attr.deadline.tv_sec =
et_sys[i].params.deadline.tv_sec;
576                         et_data->et_event_task.et_cpu_res_attr.deadline.tv_nsec =
et_sys[i].params.deadline.tv_nsec;
577
578                         et_data-
>et_event_task.et_cpu_res_attr.reserve_type.enf_mode=et_sys[i].params.reserve_type.enf
_mode;
579                         et_data-
>et_event_task.et_cpu_res_attr.reserve_type.rep_mode=et_sys[i].params.reserve_type.rep
_mode;
580                         et_data-
>et_event_task.et_cpu_res_attr.reserve_type.sch_mode=et_sys[i].params.reserve_type.sch
_mode;
581
582                                                                         /* Copy
resource names/location data */
583                         strcpy(et_data->et_event_task.et_location, "KSYSTEM");
584                         strcpy(et_data->et_event_task.et_task_type,
et_sys[i].et_rs_name);
585
586 //printf("\n>>>>>>>>>>>>>>>>>>>>SWITCH EVENT_TYPE Copying name= %s et_send= %d
loop i= %d\n", et_data->et_event_task.et_task_type, et_send, i);
587 //printf("JD()()()()()()()()()()()(>>>E_T TABULATING SYSTEM  BEFORE   CPU_LEVEL= %d
COMPUTE_TIME= %f\n",et_data->et_res_level.cpu_avail, et_time_calc(&(et_data-
>et_event_task.et_cpu_res_attr.compute_time)));
588
589                                                                         /* Update
the resource totals globally */
590                         if ((et_tab_sd = event_trace_tab(et_data)) == -1)
591
fprintf(stderr,"ERROR:URGENCY.C_SET_RK_RES_SET_URGENCY_event_trace_tab ret
%d\n",et_tab_sd);
592
593 //printf("JD()()()()()()()()()()()(>>>E_T TABULATING SYSTEM  AFTER    CPU_LEVEL= %d
COMPUTE_TIME= %f\n",et_data->et_res_level.cpu_avail, et_time_calc(&(et_data-
>et_event_task.et_cpu_res_attr.compute_time)));
594
595                                                                         /* Update
the resource totals locally */
596 //et_data->et_res_level.cpu_total
+=(float)et_time_calc(&(et_sys[i].params.compute_time));
597 //et_data->et_res_level.cpu_used
=(float)et_time_calc(&(et_sys[i].params.compute_time));
598 /*NEW*/
599 et_data_status->et_res_level.cpu_total= et_data->et_res_level.cpu_total;
600 et_data_status->et_res_level.cpu_used = et_data->et_res_level.cpu_used;
601 et_data_status->et_res_level.cpu_avail= et_data->et_res_level.cpu_avail;
```

```
602
603                         fprintf(fout,"%s      %2d       %2d   %s    %s\t  %9.3f
%9.3f %9.3f   %9.3f %9.3f  %d  %d\n", buff, et_data->et_path_len, et_data-
>et_event_task.et_trace_lev, et_data->et_event_task.et_location, et_data-
>et_event_task.et_task_type, (float)et_time_calc(&(et_data-
>et_event_task.et_cpu_res_attr.compute_time)), (float)et_time_calc(&(et_data-
>et_event_task.et_cpu_res_attr.period)), (float)et_time_calc(&(et_data-
>et_event_task.et_cpu_res_attr.deadline)), et_data->et_res_level.cpu_total, et_data-
>et_res_level.cpu_used, et_data->et_res_level.cpu_avail, et_data-
>et_res_level.percent_cpu);
604
605
606 //if (et_data_status->et_event_type != SYS_RES)
607                 if ((et_write = et_struct_reset(et_data)) == -1)
608                         fprintf(stderr,"EVENT_TRACE
ERROR:EVENT_TRACE.C_event_trace_record et_struct_reset= %d\n",et_write);
609
610                         } /* FOR ET_SYS[i] */
611                 } /* ET_SYSTEM_DATA */
612                 break;
613         case REQ_RES:
614                 strcpy(buff, "REQ_RES");
615                 break;
616         case RES_NEG:
617                 strcpy(buff, "RES_NEG");
618                 break;
619         case PATH_LN:
620                 strcpy(buff, "PATH_LN");
621                 break;
622         case RES_TYP:
623                 strcpy(buff, "RES_TYP");
624                 break;
625         case RES_ASG:
626                 strcpy(buff, "RES_ASG");
627                 break;
628         case QOS_VIO:
629                 strcpy(buff, "QOS_VIO");
630                 break;
631         case RES_RNG:
632                 strcpy(buff, "RES_RNG");
633                 break;
634         case QOS_LEV:
635                 strcpy(buff, "QOS_LEV");
636                 break;
637         case ET_NULL:
638                 strcpy(buff, "ET_NULL");
639                 break;
640         default:
641                 strcpy(buff, "ET_NULL");
642                 break;
643         }
644
645         if (!(et_pflag))
646                 fprintf(fout,"\n\n  TYPE      PATH     LEVEL   LOC       TYPE
COMPUTE_TIME    PERIOD  DEADLINE  CPU_TOTAL  CPU_USED  CPU_AVAL\n");
647
648         et_pflag = -1;
649
650         if (et_data_status->et_event_type != SYS_RES)
651                 fprintf(fout,"%s      %2d       %2d   %s    %s\t  %9.3f  %9.3f
%9.3f   %9.3f %9.3f  %d  %d\n", buff, et_data->et_path_len, et_data-
>et_event_task.et_trace_lev, et_data->et_event_task.et_location, et_data-
>et_event_task.et_task_type, (float)et_time_calc(&(et_data-
>et_event_task.et_cpu_res_attr.compute_time)), (float)et_time_calc(&(et_data-
>et_event_task.et_cpu_res_attr.period)), (float)et_time_calc(&(et_data-
>et_event_task.et_cpu_res_attr.deadline)), et_data->et_res_level.cpu_total, et_data-
>et_res_level.cpu_used, et_data->et_res_level.cpu_avail, et_data-
>et_res_level.percent_cpu);
652
653
654
```

```
655
656
657
658
659
660
661         switch (et_data_status->et_event_type)
662         {
663         case SYS_RES:
664 //              et_data->et_res_neg = et_data_status->et_res_neg; /* Update number
of resource negotiations */
665 //fprintf (stderr, "EVENT_TRACE:EVENT_TRACE.C_event_trace_record>>> RES_NEG\n");
666 //fprintf(fout, "EVENT_TRACE:EVENT_TRACE.C_event_trace_record>>> RES_NEG\n");
667                 return 0;
668                 break;
669         case REQ_RES:
670 //++my_debug;
671 //et_data->et_req_res = et_data_status->et_req_res; /* Update number of resource
reservation requests */
672 //fprintf (stderr, "EVENT_TRACE:EVENT_TRACE.C_event_trace_record>>> TYPE:
REQ_RESOURCE = %d  LOCATION: %d\n", et_data_status->et_path_len,et_data_status-
>et_event_loc);
673 //fprintf(fout, "\nCURRENT EVENT TRACE TIME: %.3f:\n",curtime);
674 //fprintf(fout, "EVENT_TRACE:EVENT_TRACE.C_event_trace_record>>> TYPE:
REQ_RESOURCE = %d  LOCATION: %d\n", et_data_status->et_path_len,et_data_status-
>et_event_loc);
675                 break;
676         case RES_NEG:
677 //et_data->et_res_neg = et_data_status->et_res_neg; /* Update number of resource
negotiations */
678 //fprintf (stderr, "EVENT_TRACE:EVENT_TRACE.C_event_trace_record>>> RES_NEG\n");
679 //fprintf(fout, "EVENT_TRACE:EVENT_TRACE.C_event_trace_record>>> RES_NEG\n");
680                 break;
681         case PATH_LN:
682 //et_data->et_path_len = +et_data_status->et_path_len;  /* Add to trace execution
path length  */
683 //fprintf (stderr, "EVENT_TRACE:EVENT_TRACE.C_event_trace_record>>> TYPE:
PATH_LENGTH = %d  LOCATION: %d\n", et_data_status->et_path_len,et_data_status-
>et_event_loc);
684 //fprintf(fout, "\nCURRENT EVENT TRACE TIME: %.3f:\n",curtime);
685 //fprintf(fout, "EVENT_TRACE:EVENT_TRACE.C_event_trace_record>>> TYPE: PATH_LENGTH
= %d  LOCATION: %d\n", et_data_status->et_path_len,et_data_status->et_event_loc);
686                 break;
687         case QOS_LEV:
688 //mcast_portnum = atoi (optarg);
689 //              et_data->et_qos_level = et_data_status->et_qos_level; /* Quality
of Service level attained/desired */
690 //fprintf (stderr, "EVENT_TRACE:EVENT_TRACE.C_event_trace_record>>> QOS_LEVEL\n");
691 //fprintf(fout, "EVENT_TRACE:EVENT_TRACE.C_event_trace_record>>> QOS_LEVEL\n");
692                 break;
693         case RES_ASG:
694 //use_internal_reporting = TRUE;
695 //              et_data->et_time_trace = et_data_status->et_time_trace; /* Updata
internal trace time clock record*/
696 //fprintf (stderr, "EVENT_TRACE:EVENT_TRACE.C_event_trace_record>>> RES_ASG\n");
697 //fprintf(fout, "EVENT_TRACE:EVENT_TRACE.C_event_trace_record>>> RES_ASING\n");
698                 break;
699         case QOS_VIO:
700 //my_repl_factor = atoi (optarg);
701 //      et_data->et_qos_violations  = et_data_status->et_qos_violations;   /*
Record violation of QoS */
702 //fprintf (stderr, "EVENT_TRACE:EVENT_TRACE.C_event_trace_record>>> QOS_VIO\n");
703 //fprintf(fout, "EVENT_TRACE:EVENT_TRACE.C_event_trace_record>>> QOS_VIO\n");
704                 break;
705         case RES_RNG:
706 //my_timeout_msec = (unsigned) SECS_TO_MSECS (atof (optarg));
707 //              et_data->et_res_reneg = et_data_status->et_res_reneg; /* Update
resource re-negotiation times/numbers */
708 //fprintf (stderr, "EVENT_TRACE:EVENT_TRACE.C_event_trace_record>>> RES_RENEG\n");
709 //fprintf(fout, "EVENT_TRACE:EVENT_TRACE.C_event_trace_record>>> RES_RENEG\n");
710                 break;
```

```
711         case RES_TYP:
712 //sys_ret =  et_get_sys_res(et_data_status);
713 //my_timeout_msec = (unsigned) SECS_TO_MSECS (atof (optarg));
714 //et_data->et_res_reneg = et_data_status->et_res_reneg; /* Update resource re-
negotiation times/numbers */
715 //fprintf (stderr, "EVENT_TRACE:EVENT_TRACE.C_event_trace_record>>> RES_RENEG\n");
716 //fprintf(fout, "EVENT_TRACE:EVENT_TRACE.C_event_trace_record>>> RES_RENEG\n");
717                 break;
718         case ET_NULL:
719
720
721
722
723
724
725 return 1;
726                 break;
727         default:
728         fprintf (stderr, "EVENT_TRACE:EVENT_TRACE.C_event_trace_record>>> TYPE:
DEFAULT  LOCATION: %d\n",et_data_status->et_event_loc);
729
730
731
732
733
734                 return -1;
735                 break;
736         }
737
738
739
740
741
742
743         if (et_data_status->et_event_type != SYS_RES)
744                 if ((et_write = et_struct_reset(et_data)) == -1)
745                         fprintf(stderr,"EVENT_TRACE
ERROR:EVENT_TRACE.C_event_trace_record et_struct_reset= %d\n",et_write);
746
747         return 0;
748
749 }; /* EVENT_TRACE_RECORD */
```

int event_trace_tab (struct event_trace_struct * *et_data_tab*)

Tabulate the system quality of service data here, and keeps dynamic totals within the et_data data structure. This function returns an int upon completion: 0=success, -1=unsuccessful.

/*!

Parameters:

struct **event_trace_struct** event trace statistics.

Returns:

Int, if successful 0, if unsuccessful -1.

Definition at line 356 of file event_trace.c.

```
357 {
358
359         int et_tab = 0;
360
361         #ifdef EVENT_TRACE_DEBUG3
362         printf("Tabulating CPU\n");
363         #endif
364
```

```
365
366         if (et_data_tab == NULL)
367                 return -1;
368
369
370
371
372         et_data_tab->et_res_level.cpu_total+=(float)et_time_calc(&(et_data_tab-
>et_event_task.et_cpu_res_attr.compute_time));
373         et_data_tab->et_res_level.cpu_used =(float)et_time_calc(&(et_data_tab-
>et_event_task.et_cpu_res_attr.compute_time));
374
375         #ifdef EVENT_TRACE_DEBUG1
376         printf("TABULATING SYSTEM  BEFORE   CPU_LEVEL= %d COMPUTE_TIME=
%f\n",et_data_tab->et_res_level.cpu_avail, et_time_calc(&(et_data_tab-
>et_event_task.et_cpu_res_attr.compute_time)));
377         #endif
378
379
380         et_data_tab->et_res_level.cpu_avail-=et_time_calc(&(et_data_tab-
>et_event_task.et_cpu_res_attr.compute_time));
381
382
383         et_tab = 0;
384
385
386
387
388
389
390
391
392
393
394
395         return(et_tab);
396
397 }; /* EVENT_TRACE_TAB */
```

event_trace.h File Reference

Include file for initial startup file in the QoS event trace.

```
#include <rk/rk.h>
#include <rk/rk_error.h>
```

Data Structures

struct et_qos_error_struct
Event Trace System Data structure for errors and misc.

struct et_sys_res
System resource data and information.

struct et_sys_task
Event Trace System Task Data.

struct et_system_status_struct
Event Trace System Data struct for system reserved resources.

struct **et_task**
struct event_trace_struct
Main event trace data/statistics.

Defines

#define TRACE_NAME_LEN 10
#define **EVENT_TRACE** 1
#define **ET_NSMS** 1000000
#define **ET_NSSEC** (1000000000L)
#define TRACE_LOC_LEN 15
#define **ET_KILOBYTE** 1024
#define TRACE_TYPE_LEN 12
#define TRACE_FILE_LEN 256
#define TRACE_TASK_LEN 10

Enumerations

enum et_qos_error_type
Quality of Service errors during event trace.

enum et_res_mode
Event trace resource reservation mode set.

enum et_type
Event trace event type.

enum **et_loc**

Event trace location.

enum **et_len**

Event trace path length location.

Functions

int **et_system_data** (int et_sys)

Retrieve all the resource kernel system reserve data, which includes the system and other resource sets. This function returns an int upon completion: 0=success, -1=unsuccessful.

int **et_struct_reset** (struct **event_trace_struct** *et_data_status)

Reset elements of the global data structure, and retain the dynamic totals within this structure. This function returns an int upon completion: 0=success, -1=unsuccessful.

float **et_time_calc** (struct timespec *et_time)

Calculate micro second data based upon the received timespec structure. This function returns an unsigned int which represents the ms time value upon successful completion, if unsuccessful returns 0.

int **event_trace_tab** (struct **event_trace_struct** *et_data_tab)

Tabulate the system quality of service data here, and keeps dynamic totals within the et_data data structure. This function returns an int upon completion: 0=success, -1=unsuccessful.

int **event_trace_init** (struct **event_trace_struct** *et_init)

Initialize all event trace params here, and allocate memory for structure This function returns an int 0 if successful, -1 if unsuccessful.

int **event_trace_record** (struct **event_trace_struct** *et_data_status)

Record QoS trace event, update data information within event trace structure. Returns updated error condition.

Variables

event_trace_struct * et_ffd

*Using typedef for event trace data information structure. \typedef struct **event_trace_struct** *event_trace_t.*

Detailed Description

Include file for initial startup file in the QoS event trace.

File : **event_trace.h** Author : John Drummond Last Modified : 2/24/02

Notes : This file is written in standard C language. The purpose of this file is to serve as a header file for the Fast Failure Detection program files that utilze structures

and defines necessary for tracing Quality of Service Events (QoSE).

Definition in file **event_trace.h**.

Define Documentation

#define ET_KILOBYTE 1024

Size of a kilobyte

Definition at line 34 of file event_trace.h.

#define ET_NSMS 1000000

Number of Nsec found within a Msec

Definition at line 24 of file event_trace.h.

#define ET_NSSEC (1000000000L)

Number of Nanoseconds within a second

Definition at line 28 of file event_trace.h.

#define EVENT_TRACE 1

Event trace session number

Definition at line 21 of file event_trace.h.

#define TRACE_FILE_LEN 256

Event trace output file length

Definition at line 40 of file event_trace.h.

#define TRACE_LOC_LEN 15

Event trace location length

Definition at line 31 of file event_trace.h.

#define TRACE_NAME_LEN 10

Name length for event trace session

Definition at line 18 of file event_trace.h.

#define TRACE_TASK_LEN 10

Length for trace task type

Definition at line 43 of file event_trace.h.

#define TRACE_TYPE_LEN 12

Length for event trace type

Definition at line 37 of file event_trace.h.

Enumeration Type Documentation

enum et_len

Event trace path length location.

This enumeration contains the following elements:

PATH_NULL -- Non specific path length

PATH_MAIN -- Main program path

PATH_INIT -- Initialization path

PATH_SET_PROC_URGENCY -- Process setup path

PATH_CREATE_RES -- Resource reservation createion path

PATH_THREAD1 -- First thread/task path

PATH_THREAD2 -- Second thread/task path

PATH_THREAD3 -- Third thread/task path

Definition at line 399 of file event_trace.h.

```
399 {PATH_NULL, PATH_MAIN, PATH_INIT, PATH_SET_PROC_URGENCY, PATH_CREATE_RES,
PATH_THREAD1, PATH_THREAD2, PATH_THREAD3};
```

enum et_loc

Event trace location.

This enumeration contains the following elements:

LOC_NULL -- Non specific location

MAIN -- Main program

INIT -- Initialization section

SET_PROC_URGENCY -- Process urgency establishment

CREATE_RES -- Resource reservation creation

THREAD1 -- First thread/task

THREAD2 -- Second thread/task

THREAD3 -- Third thread/task

Definition at line 376 of file event_trace.h.

```
376 {LOC_NULL, MAIN, INIT, SET_PROC_URGENCY, CREATE_RES, THREAD1, THREAD2, THREAD3};
```

enum et_qos_error_type

Quality of Service errors during event trace.

This enumeration contains the following elements:

ER_NULL -- Non specific error

ER_RES_REQ -- Resource Request error

ER_RES_NEG -- Resource Negotiation error

ER_RES_RENEG -- Resource Re-negotiation error

Definition at line 221 of file event_trace.h.

```
221 {ER_NULL, ER_RES_REQ, ER_RES_NEG, ER_RES_RENEG};
```

enum et_res_mode

Event trace resource reservation mode set.

This enumeration contains the following elements:

ET_HARD -- Equal to 0x1

ET_FIRM -- Equal to 0x2

ET_SOFT -- Equal to 0x4

Definition at line 311 of file event_trace.h.

```
311 {ET_HARD =0x1, ET_FIRM =0x2, ET_SOFT =0x4, };
```

enum et_type

Event trace event type.

This enumeration contains the following elements:

ET_NULL -- Non specific type

REQ_RES -- Resource Request

RES_NEG -- Resource Negotiation

PATH_LN -- Event trace path length

RES_TYP -- Reservation type

TIME_TR -- Evant trace time

QOS_VIO -- Quality of service violation

RES_RNG -- Resource re-negotiation

QOS_LEV -- Quality of service level

SYS_RES -- System resource type

Definition at line 353 of file event_trace.h.

```
353 {ET_NULL, REQ_RES, RES_NEG, PATH_LN, RES_TYP, RES_ASG, QOS_VIO, RES_RNG, QOS_LEV,
SYS_RES};
```

Variable Documentation

struct event_trace_struct* et_ffd

Using typedef for event trace data information structure. \typedef struct **event_trace_struct** *event_trace_t.

Create a type name for **event_trace_struct**

Definition at line 478 of file event_trace.h.

Referenced by et_chg_cpu_reserves(), initialize(), main(), set_process_urgency(), set_rk_thread_urgency(), and set_thread_urgency().

event_trace_system.c File Reference

System data information for the QoS event trace.

```
#include "event_trace_system.h"
```

Functions

int **et_get_sys_res** (struct **event_trace_struct** *et_data_status)

Query the system for available resources, Update data information within event trace structure. Returns updated error condition.

Detailed Description

System data information for the QoS event trace.

File : **event_trace_system.c** Author : John Drummond Last Modified On : 2/24/02

Notes : This file is written in standard C language. The purpose of the functions within this file is to allow monitoring of the system's resource data. A portion of this files functionality was leveraged from the CMU Resource Kernal system functions which monitor the /proc/ files system.

Definition in file **event_trace_system.c**.

event_trace_system.h File Reference

Include file for initial startup file in the QoS event trace.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/dir.h>
#include <getopt.h>
#include <rk/rk.h>
#include <rk/rk_error.h>
#include <rk/timespec.h>
#include <rk/posix_timers.h>
#include "event_trace.h"
```

Defines

#define **NUMRESERVES** 5
#define MAXBUFFERSIZE 50

Detailed Description

Include file for initial startup file in the QoS event trace.

File : **event_trace_system.h** Author : John Drummond Last Modified : 2/24/02

Notes : This file is written in standard C language. The purpose of this file is to serve as a header file for the Fast Failure Detection program files that utilze structures and defines necessary for tracing Quality of Service Events (QoSE).

Definition in file **event_trace_system.h**.

Define Documentation

#define MAXBUFFERSIZE 50

Size of the buffer for gather system stats

Definition at line 39 of file event_trace_system.h.

#define NUMRESERVES 5

Total number of servers for session

Definition at line 34 of file event_trace_system.h.

ffd.c File Reference

Intial app file for FFD exeriment.

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include "ffd.h"
#include "event_trace.h"
#include <rk/rk.h>
#include <rk/rk_error.h>
#include "censustaker.h"
#include "heartbeat.h"
#include "hosts.h"
#include "multicast.h"
#include "urgency.h"
```

Functions

void **initialize** (int argc, char *argv[])
\brief Initialization call from FFD.C.

int **main** (int argc, char *argv[])
Main function within the FFD.C. This begins the call to initialize for FFD experiment.

Detailed Description

Intial app file for FFD exeriment.

-*- Mode: C -*- **ffd.c** --- Author : Doug Wells Last Modified By : John Drummond Last Modified On : 2/24/02 Last Machine Used: quorum4 Update Count : 78

ffd - This program is a separate failure detector for use in systems that have a requirement for low latency detection of application failures. It has been created for the QUITE experiment on Fast Failure Detection.

There are two primary parts to this detector: a generator of heartbeat messages and a censustaker that periodically checks to see that all the other participants have been heard from. If a host fails to respond within the timeout period, that host is declared to be dead.

Currently, the program assumes that its local address is the first address returned from DNS for its primary name. If this is not the case, it will be necessary to provide a mechanism by which the proper local interface address can be specified.

The program sends out its state messages to other heartbeat actors via multicast IP. There is currently no mechanism for allowing multiple heartbeat functions to exist simultaneously. The function is limited to the local network, however, by the TTL, which is set to 1. The way to fix this is to provide a mechanism for specifying the multicast address. -Doug Wells

Definition in file **ffd.c**.

ffd.h File Reference

Include file for FFD exeriment app.

```
#include <sys/types.h>
#include <time.h>
#include "params.h"
#include "utils.h"
```

Data Structures

struct hbeat_im_alive_msg
struct hbeat_msg_hdr
struct hbeat_params_msg

Detailed Description

Include file for FFD exeriment app.

-*- Mode: C -*- **ffd.h** --- Author : Doug Wells Last Modified By : Mustafizur Rahman Last Modified On : Thu Sep 6 17:11:34 2001 Last Machine Used: quite01.camb.opengroup.org Update Count : 12

ffd.h - include file common to the Fast Failure Detection code. --Doug Wells

Definition in file **ffd.h**.

hbeatstats.c File Reference

Monitors heartbeat messages for FFD exeriment.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <limits.h>
#include "ffd.h"
#include "hosts.h"
#include "messages.h"
#include "multicast.h"
#include "params.h"
#include "urgency.h"
#include "utils.h"
```

Data Structures

struct host_entry

Detailed Description

Monitors heartbeat messages for FFD exeriment.

-*- Mode: C -*- **hbeatstats.c** --- Author : Doug Wells Last Modified By : John Drummond Last Modified On : 2/24/02 Last Machine Used: quorum4 Update Count : 84

hbeatstats - This program monitors the im_alive messages that are produced by the heartbeat generators in the QUITE experiment on Fast Failure Detection. It generates data files that can be processed to calculate various statistics, such as the jitter on the period between the messages.

This module includes a set of procedures that provide the same interface as **hosts.c**, but services the needs of the status functions. -Doug Wells

Definition in file **hbeatstats.c**.

heartbeat.c File Reference

Generates periodic heartbeat messages for FFD exeriment.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <assert.h>
#include <math.h>
#include "ffd.h"
#include "heartbeat.h"
#include "multicast.h"
#include "utils.h"
```

Detailed Description

Generates periodic heartbeat messages for FFD exeriment.

-*- Mode: C -*- **heartbeat.c** --- Author : Doug Wells Last Modified By : John Drummond Last Modified On : 2/24/02 Last Machine Used: quorum4 Update Count : 101

heartbeat - This program performs the heartbeat generator role in the heartbeat function for the QUITE experiment on Fast Failure Detection. It generates periodic messages that are send to heartbeat actors on other hosts.

Currently, the program assumes that its local address is the first address returned from DNS for its primary name. If this is not the case, it will be necessary to provide a mechanism by which the proper local interface address can be specified.

The program receives its state messages from other heartbeat actors via multicast IP. There is currently no mechanism for allowing multiple heartbeat functions to exist simultaneously. The function is limited to the local network, however, by the TTL, which is set to 1. The way to fix this is to provide a mechanism for specifying the multicast address. -Doug Wells

Definition in file **heartbeat.c**.

heartbeat.h File Reference

Include file for heartbeat generator in FFD exeriment.

Detailed Description

Include file for heartbeat generator in FFD exeriment.

-*- Mode: C -*- **heartbeat.h** --- Author : Doug Wells Last Modified By : Mustafizur Rahman Last Modified On : Thu Aug 16 14:05:43 2001 Last Machine Used: quite01.camb.opengroup.org Update Count : 11

heartbeat.h - include file to declare the top-level functions associated with the heartbeat generator function of the Fast Failure Detector. --Doug Wells

Definition in file **heartbeat.h**.

hosts.c File Reference

Host management for FFD exeriment.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include "ffd.h"
#include <pthread.h>
#include "hosts.h"
#include "multicast.h"
#include "notify.h"
```

Data Structures

struct host_entry

Detailed Description

Host management for FFD exeriment.

-*- Mode: C -*- **hosts.c** --- Author : Doug Wells Last Modified By : John Drummond Last Modified On : 2/24/02 Last Machine Used: quorum4 Update Count : 71

hosts - This set of procedures manages the state of the hosts for the heartbeat function for the QUITE experiment on Fast Failure Detection. -Doug Wells

Definition in file **hosts.c**.

hosts.h File Reference

Include file for host management in FFD exeriment.

Detailed Description

Include file for host management in FFD exeriment.

-*- Mode: C -*- **heartbeat.h** --- Author : Doug Wells Last Modified By : Mustafizur Rahman Last Modified On : Thu Aug 16 14:06:02 2001 Last Machine Used: quite01.camb.opengroup.org Update Count : 12

hosts.h - include file to declare interfaces for host functions --Doug Wells

Definition in file **hosts.h**.

iterator.c File Reference

CPU resource consumption program.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/time.h>
#include <pthread.h>
```

Data Structures

struct loop_arg_t

Detailed Description

CPU resource consumption program.

-*- Mode: C -*- **iterator.c** --- Author : Mustafizur Rahman (rahman@jajabor.com) Created On : Thu Feb 22 20:29:43 2001 Last Modified By : John Drummond Last Modified On : 2/24/02 Last Machine Used: quorum4 Update Count : 306

This simple program keeping CPU busy by adding counters.

Definition in file **iterator.c**.

longlong.h File Reference

Establishes defs for 32-bit & 64-bit arithmetic functions within RK.

Detailed Description

Establishes defs for 32-bit & 64-bit arithmetic functions within RK.

Definition in file **longlong.h**.

Define Documentation

#define __umulsidi3(u, v)

Value:

```
({DIunion __w;                                                  \
    umul_ppmm (__w.s.high, __w.s.low, u, v);                    \
    __w.ll; })
```

Definition at line 1175 of file longlong.h.

#define add_ssaaaa(sh, sl, ah, al, bh, bl)

Value:

```
do {                                                            \
    USItype __x;                                                \
    __x = (al) + (bl);                                          \
    (sh) = (ah) + (bh) + (__x < (al));                          \
    (sl) = __x;                                                 \
  } while (0)
```

Definition at line 1129 of file longlong.h.

#define count_leading_zeros(count, x)

Value:

```
do {                                                            \
    USItype __xr = (x);                                         \
    USItype __a;                                                \
                                                                \
    if (SI_TYPE_SIZE <= 32)                                     \
      {                                                         \
        __a = __xr < ((USItype)1<<2*__BITS4)                    \
          ? (__xr < ((USItype)1<<__BITS4) ? 0 : __BITS4)        \
          : (__xr < ((USItype)1<<3*__BITS4) ?  2*__BITS4 : 3*__BITS4);  \
      }                                                         \
    else                                                        \
      {                                                         \
        for (__a = SI_TYPE_SIZE - 8; __a > 0; __a -= 8)         \
          if (((__xr >> __a) & 0xff) != 0)                      \
            break;                                              \
      }                                                         \
                                                                \
    (count) = SI_TYPE_SIZE - (__clz_tab[__xr >> __a] + __a);    \
  } while (0)
```

Definition at line 1238 of file longlong.h.

#define sub_ddmmss(sh, sl, ah, al, bh, bl)

Value:

```
do {                                                            \
    USItype __x;                                                \
```

```
    __x = (al) - (bl);                                                  \
    (sh) = (ah) - (bh) - (__x > (al));                                  \
    (sl) = __x;                                                         \
  } while (0)
```

Definition at line 1139 of file longlong.h.

#define umul_ppmm(w1, w0, u, v)

Value:

```
do {                                                                    \
    USItype __x0, __x1, __x2, __x3;                                     \
    USItype __ul, __vl, __uh, __vh;                                     \
                                                                        \
    __ul = __ll_lowpart (u);                                            \
    __uh = __ll_highpart (u);                                           \
    __vl = __ll_lowpart (v);                                            \
    __vh = __ll_highpart (v);                                           \
                                                                        \
    __x0 = (USItype) __ul * __vl;                                       \
    __x1 = (USItype) __ul * __vh;                                       \
    __x2 = (USItype) __uh * __vl;                                       \
    __x3 = (USItype) __uh * __vh;                                       \
                                                                        \
    __x1 += __ll_highpart (__x0);               \
    __x1 += __x2;                               \
    if (__x1 < __x2)                                        \
      __x3 += __ll_B;                   \
                                                                        \
    (w1) = __x3 + __ll_highpart (__x1);                                 \
    (w0) = __ll_lowpart (__x1) * __ll_B + __ll_lowpart (__x0);          \
  } while (0)
```

Definition at line 1149 of file longlong.h.

messages.c File Reference

Incoming message decoder file for FFD exeriment.

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <fcntl.h>
#include <limits.h>
#include "ffd.h"
#include "messages.h"
#include "hosts.h"
```

Detailed Description

Incoming message decoder file for FFD exeriment.

-*- Mode: C -*- **messages.c** --- Author : Doug Wells Last Modified By : Mustafizur Rahman Last Modified On : Thu Aug 16 14:06:26 2001 Last Machine Used: quite01.camb.opengroup.org Update Count : 90

messages - This program decodes the incoming messages for the the census taker role in the heartbeat function for the QUITE experiment on Fast Failure Detection. - Doug Wells

Definition in file **messages.c**.

messages.h File Reference

Include file for incoming message decoder in FFD exeriment.

Detailed Description

Include file for incoming message decoder in FFD exeriment.

-*- Mode: C -*- **messages.h** --- Author : Doug Wells Last Modified By : Mustafizur Rahman Last Modified On : Thu Aug 16 14:06:19 2001 Last Machine Used: quite01.camb.opengroup.org Update Count : 10

messages.h - include file to declare top-level functions for the message decoder in the Fast Failure Detector. --Doug Wells

Definition in file **messages.h**.

misc.c File Reference

Miscellaneous RK system call functions used in Linux Resource Kernel.

```
#include <linux/config.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/errno.h>
#include <linux/sched.h>
#include <rk/rk_linux.h>
#include <rk/rk.h>
#include <rk/posix_timers.h>
#include <linux/timer.h>
```

Functions

void **rk_timer_destroy** (rk_timer_t)

Timer management function.

void **rk_timer_remove** (rk_timer_t)

Disarms a potentially armed timer; call this before destroying unless you know that the timer is not armed.

Detailed Description

Miscellaneous RK system call functions used in Linux Resource Kernel.

misc.c: Miscellaneous RK functions

Definition in file **misc.c**.

multicast.c File Reference

Multicast socket manager file for FFD exeriment.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <assert.h>
#include <fcntl.h>
#include "ffd.h"
#include "multicast.h"
```

Detailed Description

Multicast socket manager file for FFD exeriment.

-*- Mode: C -*- **multicast.c** --- Author : Doug Wells Last Modified By : Mustafizur Rahman Last Modified On : Thu Aug 16 14:06:57 2001 Last Machine Used: quite01.camb.opengroup.org Update Count : 2

multicast - This module contains procdures to manage multicast sockets for the Fast Failure Detector.

Currently, the program assumes that its local address is the first address returned from DNS for its primary name. If this is not the case, it will be necessary to provide a mechanism by which the proper local interface address can be specified. -Doug Wells

Definition in file **multicast.c**.

multicast.h File Reference

Include file for multicast socket manager in FFD exeriment.

Detailed Description

Include file for multicast socket manager in FFD exeriment.

-*- Mode: C -*- **multicast.h** --- Author : Doug Wells Last Modified By : Mustafizur Rahman Last Modified On : Thu Aug 16 14:06:51 2001 Last Machine Used: quite01.camb.opengroup.org Update Count : 2

multicast.h - include file to declare procedures for managing multicast sockets for the Fast Failure Detector. --Doug Wells

Definition in file **multicast.h**.

mutex.c File Reference

Provides mutex control & priority inheritance support for RK.

```
#include <linux/config.h>
#include <linux/sched.h>
#include <rk/rk_linux.h>
#include <rk/rk.h>
#include <linux/list.h>
```

Data Structures

struct **mutex**
struct mutex_list
struct task_list

Functions

void **add_owned_mutex_pq** (mutex_list *mtxl, struct list_head *head)

Add the mutex to the owned mutex of the process. head is the pointer from the process to the owned mutexes.

void **switch_resource_set** (struct task_struct *proc, struct rk_resource_set *prev, struct rk_resource_set *next)

This function switches from one resource_set to another and changes the rs_proc_list of the rs. Note that the resource set list of the process is not changed though. This is because this switch is considered a temporal assignment.

void **rt_adjbaseprio** (struct task_struct *proc, int policy, int newbase)

Adjusts the base priority of a process, ensuring that priority queues are adjusted and that any priority inheritance/uninheritance needed is performed.

Detailed Description

Provides mutex control & priority inheritance support for RK.

mutex.c: mutex and priority inheritance support

Definition in file **mutex.c**.

Function Documentation

void rt_adjbaseprio (struct task_struct * *proc*, int *policy*, int *newbase*)

Adjusts the base priority of a process, ensuring that priority queues are adjusted and that any priority inheritance/uninheritance needed is performed.

for reserve inheritance the strategy is as follows: adjust priority OnDepletion (newbase < current_base) if base priority is higher return to base resource set if OWNER_OF_MUTEX: check if the highest priority among the blocked processes is higher than mine. If so inherit make sure that you do not inherit the same reserve that is being adjusted (it could happen that the blocked process has not being adjusted yet). if BLOCKED_ON_MUTEX: reposition itself in the mutex priority queue. if WAS HIGHEST blocked on this mutex reposition the mutex in the priority queue of the owner's owned mutexes. NOTE THAT THE PROPAGATION OF THE PRIORITY ON THE DEPLETION CASE WILL HAPPEN WHEN EACH PROCESS ATTACHED TO THIS RESERVE IS DEPLETED. OnReplenishment (newbase > current_base) if BLOCKED_ON_MUTEX: reposition itself in the mutex priority queue. if HIGHEST blocked on this mutex reposition the mutex in the priority queue of owner's owned mutexes. if mutex is HIGHEST owned mutex propagate rs to this process. propagate this rs through the chain of blocked processes on mutex. NOTE THAT THE PROPAGATION OF THE REPLENISHMENT OF THIS RS NEEDS TO BE DONE AT THIS TIME.

Preconditions: The status of the process that is being adjusted is as follows: rk_resource_set has: the eligible resource set with the highest priority from the queue of resource sets. Note that the inherited resource set is not added to this queue given that on every adjustment we need to re-inherit any way. the newbase is the priority that corresponds to the current resource_set

We need to represent the blocking of a process as a low priority so we can use this same function to search for a reserve to inherit. After we try to inherit the reserve if the priority is still the 'blocking' priority (we need to use a negative value) we need to return it to a zero value.

Postconditions: The process currently being adjusted should be attached to it's own resource_set to receive the replenishment event, and to any inherited resource set to receive the depletion event. It is possible to be attached to a inherited resource set after a depletion if the inherited event has a higher depleted priority that our own priority. This should be taken into account in the new reserve and resource set schedule function for the multi-reserve/resource set scheduler

The resulting priority could be negative if the 'blocking' priority is chosen to be so. In that case the calling function is responsible of returning that priority to positive and actually blocking the process.

Definition at line 1076 of file mutex.c.

Referenced by rk_timer_destroy().

```
1077 {
1078     int depletion = (proc->rt_priority > newbase);
1079 #ifdef RK_MUTEX_DEBUG_INHERIT
1080     printk("rt_adjbaseprio pid(%d), policy(%d), priority(%d) rs(%x)\n",
1081            proc->pid, policy, newbase,
1082            (unsigned int) proc->rk_resource_set);
1083 #endif
1084
1085 #ifdef RK_MUTEX_DEBUG_1
1086     printk("rt_adjbaseprio pid(%d), policy(%d), priority(%d) rs(%x)\n",
1087            proc->pid, policy, newbase,
1088            (unsigned int) proc->rk_resource_set);
1089 #endif
1090
1091     proc->policy = policy;
```

```
1092     proc->rt_priority = newbase;
1093
1094     /* if the resource set being adjusted is the current base then
1095      * adjust it
1096      */
1097
1098     if (proc->rk_base_resource_set == proc->rk_resource_set) {
1099         proc->rt_base_policy = policy;
1100         proc->rt_base_priority = newbase;
1101     }
1102
1103
1104     if (depletion) {
1105 #ifdef RK_MUTEX_DEBUG_3
1106         printk("adjbase: before try_reinh\n");
1107 #endif
1108         try_reinheritance(proc);
1109     }
1110
1111
1112     /* check if this process is blocked waiting for a mutex */
1113 #ifdef RK_MUTEX_DEBUG_3
1114     printk("adjbase: before blocking_chain...\n");
1115 #endif
1116
1117 #if RSV_PROPAGATION
1118     // blocking_chain_priority_propagation(proc, !depletion);
1119     blocking_chain_priority_propagation(proc, 1);
1120 #endif
1121
1122 #ifdef RK_MUTEX_DEBUG_1
1123     printk
1124         ("rt_adjbaseprio pid(%d)i, policy(%d), priority(%d), rs(%x) done\n",
1125          proc->pid, (int) proc->policy, (int) proc->rt_priority,
1126          (unsigned int) proc->rk_resource_set);
1127 #endif
1128 #ifdef RK_MUTEX_DEBUG_INHERIT
1129     printk
1130         ("rt_adjbaseprio pid(%d)i, policy(%d), priority(%d), rs(%x) done\n",
1131          proc->pid, (int) proc->policy, (int) proc->rt_priority,
1132          (unsigned int) proc->rk_resource_set);
1133 #endif
1134 }
```

net_reserve.c File Reference

Established network resource reserves for the Linux Resource Kernel.

```
#include <rk/rk_linux.h>
#include <rk/rk_error.h>
#include <rk/rk.h>
#include <rk/timespec.h>
#include <linux/time.h>
#include <asm/processor.h>
```

Detailed Description

Established network resource reserves for the Linux Resource Kernel.

Road map of net reserve:

1. create a net reserve and counting a total number of bytes sent through it
2. perform accounting based on C/T model
3. introduce the enforcement for net reserves

Definition in file **net_reserve.c**.

notify.c File Reference

Communication management of hosts state in FFD exeriment.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include "ffd.h"
#include "notify.h"
#include "params.h"
```

Detailed Description

Communication management of hosts state in FFD exeriment.

-*- Mode: C -*- **notify.c** --- Author : Doug Wells Last Modified By : Mustafizur Rahman Last Modified On : Thu Sep 27 11:02:19 2001 Last Machine Used: quite01.camb.opengroup.org Update Count : 138

notify - This set of procedures manages the communication of host state to external agents. -Doug Wells

Definition in file **notify.c**.

notify.h File Reference

Include file for communication of hosts state in FFD exeriment.

Detailed Description

Include file for communication of hosts state in FFD exeriment.

-*- Mode: C -*- **notify.h** --- Author : Doug Wells Last Modified By : Mustafizur Rahman Last Modified On : Thu Aug 16 14:07:05 2001 Last Machine Used: quite01.camb.opengroup.org Update Count : 3

notify.h - include file to declare interfaces for procedures to communicate host state to external agents. --Doug Wells

Definition in file **notify.h**.

params.c File Reference

Intial settings for heartbeat & debug parameters.

```
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>
#include "ffd.h"
#include "urgency.h"
#include "event_trace.h"
```

Functions

void propagate_current_params (void)
void set_default_params (void)

Detailed Description

Intial settings for heartbeat & debug parameters.

-*- Mode: C -*- **params.c** --- Author : Doug Wells Last Modified By : John Drummond Last Modified On : 2/24/02 Last Machine Used: quorum4 Update Count : 2

Definition in file **params.c**.

Function Documentation

void propagate_current_params (void)

Setting heartbeat replication factor & timeout/reporting periods.

Definition at line 120 of file params.c.

Referenced by set_default_params().

```
121 {
122
123         #ifdef EVENT_TRACE_DEBUG3
124         printf("Propagate_current_params\n");
125         #endif
126
127
128         if (hbeat_debug != NULL_DEBUG)
129                 set_debug_param (hbeat_debug);
130
131         if (hbeat_timeout_period_msec != NULL_TIMEOUT_PERIOD)
132         {
133                 double  new_timeout_period = MSECS_TO_SECS
(hbeat_timeout_period_msec);
134
135                 if (new_timeout_period != timeout_period)
136                 {
137                         fprintf (stderr,
138                           "Changing timeout from %.3f to %.3f secs\n",
139                           timeout_period, new_timeout_period);
140                         timeout_period = new_timeout_period;
```

```
141                 }
142         }
143
144         if (hbeat_replication_factor != NULL_REPLICATION_FACTOR)
145         {
146                 if (hbeat_replication_factor != replication_factor)
147                 {
148                         fprintf (stderr,
149                           "Changing replication_factor from %d to %ld\n",
150                           replication_factor, hbeat_replication_factor);
151                         replication_factor = hbeat_replication_factor;
152                 }
153         }
154
155         if (hbeat_report_period_msec != NULL_REPORT_PERIOD)
156         {
157                 double  new_report_period = MSECS_TO_SECS
(hbeat_report_period_msec);
158
159                 if (new_report_period != report_period)
160                 {
161                         fprintf (stderr,
162                           "Changing report_period from %.3f to %.3f secs\n",
163                           report_period, new_report_period);
164                         report_period = new_report_period;
165                 }
166         }
167
168
169         assert (timeout_period != 0);
170         assert (report_period != 0);
171         assert (replication_factor != 0);
172
173         heartbeat_period = timeout_period / replication_factor;
174         ++parameter_changes;
175 }
```

void set_default_params (void)

Setting heartbeat & debug parameters.

Definition at line 180 of file params.c.

References propagate_current_params().

Referenced by initialize().

```
181 {
182
183         #ifdef EVENT_TRACE_DEBUG3
184         printf("Set_default_params\n");
185         #endif
186
187         hbeat_debug = NULL_DEBUG;
188         hbeat_timeout_period_msec = NULL_TIMEOUT_PERIOD;
189         hbeat_replication_factor = NULL_REPLICATION_FACTOR;
190         hbeat_report_period_msec = NULL_REPORT_PERIOD;
191
192         debug = DEFAULT_DEBUG;
193         timeout_period = DEFAULT_TIMEOUT;
194         replication_factor = DEFAULT_REPLICATION;
195         report_period = DEFAULT_REPORT_PERIOD;
196
197         if (getenv ("DEBUGFFD") != NULL)
198                 ++debug;
199
```

```
200         propagate_current_params ();
201 }
```

params.h File Reference

Include file for heartbeat settings in FFD experiment.

Functions

void set_default_params (void)

Detailed Description

Include file for heartbeat settings in FFD experiment.

-*- Mode: C -*- **params.h** --- Author : Doug Wells Last Modified By : Mustafizur Rahman Last Modified On : Thu Aug 16 14:07:29 2001 Last Machine Used: quite01.camb.opengroup.org Update Count : 3

params.h - include file to declare functions and variables for parameters. --Doug Wells

Definition in file **params.h**.

Function Documentation

void set_default_params (void)

Setting heartbeat & debug parameters.

Definition at line 180 of file params.c.

```
181 {
182
183         #ifdef EVENT_TRACE_DEBUG3
184         printf("Set_default_params\n");
185         #endif
186
187         hbeat_debug = NULL_DEBUG;
188         hbeat_timeout_period_msec = NULL_TIMEOUT_PERIOD;
189         hbeat_replication_factor = NULL_REPLICATION_FACTOR;
190         hbeat_report_period_msec = NULL_REPORT_PERIOD;
191
192         debug = DEFAULT_DEBUG;
193         timeout_period = DEFAULT_TIMEOUT;
194         replication_factor = DEFAULT_REPLICATION;
195         report_period = DEFAULT_REPORT_PERIOD;
196
197         if (getenv ("DEBUGFFD") != NULL)
198                 ++debug;
199
200         propagate_current_params ();
201 }
```

posix_timers.h File Reference

Sets up includes for posix timer use in the Linux Resource Kernel.

```
#include <signal.h>
```

Data Structures

struct itimerspec

Timer period & timer experation structures.

struct posix_timer

Posix timer specific structures.

Defines

#define CLOCK_REALTIME 0

Detailed Description

Sets up includes for posix timer use in the Linux Resource Kernel.

Definition in file **posix_timers.h**.

Define Documentation

#define CLOCK_REALTIME 0

Defines the realtime clock flag

Definition at line 36 of file posix_timers.h.

Referenced by et_get_sys_res(), and rk_replenish_timer_create().

ppc_timer.c File Reference

High-res timer support for the PowerPC using Linux Resource Kernel.

```
#include <linux/config.h>
#include <linux/kernel.h>
#include <asm/io.h>
#include <asm/timex.h>
#include <rk/rk_linux.h>
#include <rk/rk.h>
```

Functions

__inline__ unsigned int **get_dec** (void)

Accessor functions for the decrementer register.

void **rk_update_hw_timer** (struct rk_timer *tmr)

Set the decrementer to go off when the first timer expires.

Detailed Description

High-res timer support for the PowerPC using Linux Resource Kernel.

ppc_timer.c: PowerPC high resolution timer support

Definition in file **ppc_timer.c**.

Variable Documentation

char* tmrtype[]

Initial value:

```
{
    "TMR_NULL",
    "TMR_REPLENISH_RSV",
    "TMR_PERIOD_START",
    "TMR_NEXT_PERIOD",
    "TMR_POSIX",
    "TMR_JIFFY",
    "TMR_ENFORCE"
```

```
}
```

Definition at line 50 of file ppc_timer.c.

reserve.c File Reference

Provides generic reserve handling functions for the Linux Resource Kernel.

```
#include <rk/rk_linux.h>
#include <rk/rk_error.h>
#include <rk/rk.h>
#include <asm/uaccess.h>
```

Functions

void **rk_reserve_init** (void)

Initialize resource kernel functions reserves.

void **rk_reserve_cleanup** (void)

Clean up resource kernal reserve elements.

Detailed Description

Provides generic reserve handling functions for the Linux Resource Kernel.

reserve.c: generic reserve handling functions

Electrical and Computer Engineering, and Computer Science Carnegie Mellon University Pittsburgh PA 15213-3890

or via email to `raj@ece.cmu.edu`

any improvements or extensions that they make and grant Carnegie Mellon the rights to redistribute these changes.

Definition in file **reserve.c**.

resource_set.c File Reference

Resource setup file for Linux Resource Kernel.

```
#include <rk/rk_linux.h>
#include <rk/rk_error.h>
#include <rk/rk.h>
#include <rk/timespec.h>
#include <linux/time.h>
#include <asm/processor.h>
#include <asm/uaccess.h>
```

Functions

void **rk_resource_set_schedule** (struct rs_proc_list *rs_proc, unsigned int arg)

Reschedule the eligible resource set with highest priority (top of the rs_list). If no eligible resource set is available, halt the task. Otherwise, update the scheduling parameter of the task.

void **int_rk_resource_set_detach_process** (struct rs_proc_list *rs_proc, rk_resource_set_t rs)

Detach resource set from the specified rs_proc. Then choose the next eligible resource set to run. If no eligible resource set is available, halt the process.

void **rk_resource_set_adjust_accounting** (void)

Update the accounting of the current task if its scheduling parameter changes.

rk_reserve_t **rk_resource_set_update_cpu** (rk_resource_set_t rs)

Update the current cpu reserve of the resource set. This function will choose the eligible cpu reserve (~RSV_DEPLETED & RSV_ACTIVE) and set it to rs_cpu of the resources_set. It returns the chosen cpu reserve or NULL_RESERVE if no eligible reserve is available.

void **rk_resource_set_check_default_rs** (struct rs_proc_list *rs_proc, unsigned int arg)

Check and activate/deactivate the default resource set to the task. If all resource sets attached to the task have HARD enforcement, we should unlink the default resource set from the task. Otherwise, the default resource is automatically linked.

rk_resource_set_t **rk_resource_set_create** (char *name)

Create the resource kernel resource set.

void **rk_resource_set_destroy_reserves** (struct list_head *rsv_list)

***rk_resource_set_destroy_reserves**(struct list_head *rsv_list) called from rk_resource_set_destroy to destroy all the reserves on the list specified by ``rsv_list''.*

int **rk_resource_set_attach_reserve** (rk_resource_set_t rs, rk_reserve_t rsv)

Attach reserves to the previously established resource set.

void **rk_resource_set_detach_reserve** (rk_resource_set_t rs, rk_reserve_t rsv)

Detach reserves from the established resource set.

void **rk_resource_set_update_sch_mode** (rk_resource_set_t rs)

Update the sch_mode of the resource set. If all cpu reserves attached to the resource set have HARD enforcement, the sch_mode is HARD. Otherwise, it is SOFT. I haven't considered the FIRM TYPE sch_mode yet.

void **rk_resource_set_adjust_expect_accounting** (void)

Update the accounting of the expected current task if its scheduling parameter changes.

void **rk_resource_set_tsk_delete_rs** (rk_resource_set_t rs, struct task_struct *tsk)

Delete a resource set from the list in the task.

Variables

int **dbgvar** = 0

**JD*/.*

Detailed Description

Resource setup file for Linux Resource Kernel.

resource_set.c: code to manage resource sets

Real-Time and Multimedia Systems Laboratory Attn: Prof. Raj Rajkumar Electrical and Computer Engineering, and Computer Science Carnegie Mellon University Pittsburgh PA 15213-3890

or via email to `raj@ece.cmu.edu`

any improvements or extensions that they make and grant Carnegie Mellon the rights to redistribute these changes.

Definition in file **resource_set.c**.

rk.h File Reference

Sets up structures for the RK system within the Linux Resource Kernel.

```
#include <time.h>
```

Data Structures

struct cpu_reserve_attr
struct disk_reserve_attr
struct et_trace_level
struct net_reserve_attr
struct rk_et_event_struct
struct rk_et_trace_struct
struct rk_reserve_param

Functions

rk_resource_set_t **rk_resource_set_create** (char *)
Create the resource kernel resource set.

Detailed Description

Sets up structures for the RK system within the Linux Resource Kernel.

Definition in file **rk.h**.

rk_error.h File Reference

Defines RK success/failure values for use in the Linux Resource Kernel.

Detailed Description

Defines RK success/failure values for use in the Linux Resource Kernel.

Definition in file **rk_error.h**.

rk_init.c File Reference

This file provides the initialization for the Resource Kernel.

```
#include <rk/rk_linux.h>
#include <rk/rk.h>
```

Functions

void **rk_reserve_init** (void)
Initialize resource kernel functions reserves.

void **rk_reserve_cleanup** (void)
Clean up resource kernal reserve elements.

void **disk_reserve_init** (void)
Initialize DISK reserves.

rk_reserve_t **cpu_reserve_dummy_create** (rk_resource_set_t rs, time_t duration)
The cpu reserve for measurement only. This is for default and idle resource sets. No need to apply admission control to these reserves. We apply the 100% capacity reserve. The parameter duration is the measurement granularity we want to keep track the cpu usage.

Detailed Description

This file provides the initialization for the Resource Kernel.

rk_init.c: Initialization code for RK

CARNEGIE MELLON ALLOWS FREE USE OF THIS SOFTWARE IN ITS "AS IS" CONDITION. CARNEGIE MELLON DISCLAIMS ANY LIABILITY OF ANY KIND FOR ANY DAMAGES WHATSOEVER RESULTING FROM THE USE OF THIS SOFTWARE.

Carnegie Mellon requests users of this software to return to

Real-Time and Multimedia Systems Laboratory Attn: Prof. Raj Rajkumar Electrical and Computer Engineering, and Computer Science Carnegie Mellon University Pittsburgh PA 15213-3890

or via email to `raj@ece.cmu.edu`

any improvements or extensions that they make and grant Carnegie Mellon the rights to redistribute these changes.

Definition in file **rk_init.c**.

rk_isr.c File Reference

This file sets up interrupt hooks for the Linux Resource Kernel.

```
#include <rk/rk_linux.h>
#include <rk/rk_error.h>
#include <rk/rk.h>
#include <asm/system.h>
#include <asm/io.h>
```

Functions

void **cpu_reserve_sched_disable** (struct rs_proc_list *rs_proc, unsigned int arg)
Disable scheduler from making process eligible for execution.

asmlinkage int **rk_ret_with_reschedule** (struct pt_regs regs)
For "rk_ret_with_reschedule_hook".

Detailed Description

This file sets up interrupt hooks for the Linux Resource Kernel.

rk_isr.c: interrupt hooks

Real-Time and Multimedia Systems Laboratory Attn: Prof. Raj Rajkumar Electrical and Computer Engineering, and Computer Science Carnegie Mellon University Pittsburgh PA 15213-3890

or via email to `raj@ece.cmu.edu`

any improvements or extensions that they make and grant Carnegie Mellon the rights to redistribute these changes.

Definition in file **rk_isr.c**.

rk_linux.h File Reference

This file sets up linux hooks for use in the Linux Resource Kernel.

```
#include <linux/kernel.h>
#include <linux/types.h>
#include <linux/list.h>
#include <linux/sched.h>
#include <linux/wait.h>
#include <linux/slab.h>
#include <linux/param.h>
#include <asm/spinlock.h>
#include <rk/rk.h>
```

Data Structures

struct rs_proc_list

Detailed Description

This file sets up linux hooks for use in the Linux Resource Kernel.

Definition in file **rk_linux.h**.

rk_procfs.c File Reference

Establishes interface for Linux procfs within the Linux Resource Kernel.

```
#include <linux/init.h>
#include <linux/proc_fs.h>
#include <linux/stat.h>
#include <rk/rk_linux.h>
#include <rk/rk.h>
```

Data Structures

struct proc_process_list

Detailed Description

Establishes interface for Linux procfs within the Linux Resource Kernel.

rk_procfs.c (for Linux): implements the interface for procfs. shows the information on resource sets and reserves.

any improvements or extensions that they make and grant Carnegie Mellon the rights to redistribute these changes.

Definition in file **rk_procfs.c**.

rk_sched.c File Reference

This file implements scheduling hooks for the Linux Resource Kernel.

```
#include <rk/rk_linux.h>
#include <rk/rk.h>
```

Detailed Description

This file implements scheduling hooks for the Linux Resource Kernel.

rk_sched.c: scheduler hooks

Definition in file **rk_sched.c**.

rt_process.c File Reference

Sets up periodic thread & misc R/T support for Linux Resource Kernel.

```
#include <rk/rk_linux.h>
#include <rk/rk_error.h>
#include <rk/rk.h>
#include <rk/timespec.h>
#include <linux/time.h>
#include <asm/processor.h>
#include <linux/errno.h>
#include <linux/sched.h>
#include <linux/wait.h>
#include <asm/uaccess.h>
```

Detailed Description

Sets up periodic thread & misc R/T support for Linux Resource Kernel.

rt_process.c: periodic thread and miscellaneous real-time support

Definition in file **rt_process.c**.

timer.c File Reference

Provides an architecture independent high resolution timer support for RK.

```
#include <rk/rk_linux.h>
#include <rk/rk_error.h>
#include <rk/rk.h>
#include <rk/posix_timers.h>
#include <rk/timespec.h>
#include <linux/timer.h>
#include <linux/sched.h>
#include <asm/semaphore.h>
#include <errno.h>
#include <asm/uaccess.h>
#include <linux/timex.h>
#include <asm/io.h>
```

Functions

rk_timer_t **rk_timer_create** (void)
Timer management function.

void **rk_timer_destroy** (rk_timer_t tmr)
Timer management function.

void **rk_timer_add** (rk_timer_t tmr)
Timer management function Arms an already created timer; caller should hold the RK lock.

void **rk_timer_remove** (rk_timer_t tmr)
Disarms a potentially armed timer; call this before destroying unless you know that the timer is not armed.

void **rk_replenish_reserve** (rk_timer_t tmr)
***rk_replenish_reserve**(rk_timer_t tmr), called with lock.*

void **rk_start_reserve** (rk_timer_t tmr)
This function is needed for the following reason. Suppose that the user creates a resource set, and attaches a process to the resource set BEFORE creating the CPU reserve inside the resource set. In that case, we need to start running the process as soon as the reserve is eligible to execute.

void **rk_replenish_timer_create** (rk_reserve_t rsv, struct timespec start)
Create a timer to replenish a reserve, add it to the queue. Set a linux timer if necessary. ***rk_replenish_timer_create()***

Detailed Description

Provides an architecture independent high resolution timer support for RK.

timer.c: Architecture independent high resolution timer support.

Definition in file **timer.c**.

Function Documentation

void rk_start_reserve (rk_timer_t *tmr*)

This function is needed for the following reason. Suppose that the user creates a resource set, and attaches a process to the resource set BEFORE creating the CPU reserve inside the resource set. In that case, we need to start running the process as soon as the reserve is eligible to execute.

But this also means that the reserve will not be attached to the CPU resource set until later.

rk_start_reserve (rk_timer_t tmr) is called with lock

Definition at line 286 of file timer.c.

References cpu_reserve_delete(), rk_replenish_reserve(), rk_resource_set_attach_reserve(), rk_timer_add(), and rk_timer_destroy().

Referenced by rk_replenish_timer_create().

```
287 {
288     rk_reserve_t rsv = tmr->tmr_rsv;
289     cpu_tick_data_t period;
290     rk_resource_set_t rs = rsv->rsv_rs;
291 
292     if (tmr->tmr_type != TMR_REPLENISH_RSV) {
293         /* something wrong */
294         rk_timer_destroy(tmr);
295     }
296 
297     /* enable cpu reserve  */
298     cpu_reserve_start(rsv, &tmr->tmr_expire);
299 
300     /* attach reserve to resource set */
301     if (rk_resource_set_attach_reserve(rs, rsv) != RK_SUCCESS) {
302         /* another reserve attached in the meantime? Delete reserve */
303         cpu_reserve_delete(rs);
304 
305         /* timer essentially commits suicide */
306         rk_timer_destroy(tmr);
307         return;
308     }
309 
310     /* replenish it, and set timer expiration "ticks" later */
311     (rsv->rsv_ops->replenish) (rsv, &period, &tmr->tmr_expire);
312     tmr->tmr_expire += period;
313 
314 #ifdef DEBUG_RK
315     printk("rk_start_reserve: next expire(%lu usec)\n",
316            TICK2USEC(&tmr->tmr_expire));
317 #endif                          /* DEBUG_RK */
318 
319     /* set next timer: this IS needed */
320     tmr->tmr_handler = rk_replenish_reserve;
321 
322     /* add timer to queue */
323     rk_timer_add(tmr);
324 }
```

timespec.h File Reference

Sets up time functions for utilization in the Linux Resource Kernel.

Defines

#define **NANOSEC_PER_SEC** (1000000000L)

Detailed Description

Sets up time functions for utilization in the Linux Resource Kernel.

Definition in file **timespec.h**.

Define Documentation

#define nano2timespec(ts, nanos)

Value:

```
do {\
        ts.tv_sec=nanos/1000000000; \
        ts.tv_nsec=nanos-ts.tv_sec; \
} while (0)
```

Definition at line 100 of file timespec.h.

#define NANOSEC_PER_SEC (1000000000L)

Nanoseconds within a second in Long

Definition at line 11 of file timespec.h.

#define timespec_add(result, addend)

Value:

```
do {  \
        (result).tv_nsec += (addend).tv_nsec; \
        (result).tv_sec += (addend).tv_sec; \
        if ((result).tv_nsec >= NANOSEC_PER_SEC) { \
                (result).tv_nsec -= NANOSEC_PER_SEC; \
                (result).tv_sec++; \
        } \
} while (0)
```

Definition at line 22 of file timespec.h.

#define timespec_add_nsec(result, nanos)

Value:

```
do { \
        if (((result).tv_nsec += (nanos)) >= NANOSEC_PER_SEC) { \
                (result).tv_nsec -= NANOSEC_PER_SEC; \
                (result).tv_sec++; \
        } \
} while (0)
```

Definition at line 15 of file timespec.h.

#define timespec_cmp(time1, time2)

Value:

```
(((time1).tv_sec < (time2).tv_sec) || \
         (((time1).tv_sec == (time2).tv_sec) && \
         ((time1).tv_nsec <= (time2).tv_nsec)))
```

Definition at line 47 of file timespec.h.

#define timespec_eq(time1, time2)

Value:

```
(((time1).tv_sec == (time2).tv_sec) &&  \
        ((time1).tv_nsec == (time2).tv_nsec))
```

Definition at line 72 of file timespec.h.

#define timespec_ge(time1, time2)

Value:

```
(((time1).tv_sec > (time2).tv_sec) || \
         (((time1).tv_sec == (time2).tv_sec) && \
         ((time1).tv_nsec >= (time2).tv_nsec)))
```

Definition at line 52 of file timespec.h.

#define timespec_gt(time1, time2)

Value:

```
(((time1).tv_sec > (time2).tv_sec) || \
         (((time1).tv_sec == (time2).tv_sec) && \
         ((time1).tv_nsec > (time2).tv_nsec)))
```

Definition at line 57 of file timespec.h.

#define timespec_le(time1, time2)

Value:

```
(((time1).tv_sec < (time2).tv_sec) || \
         (((time1).tv_sec == (time2).tv_sec) && \
         ((time1).tv_nsec <= (time2).tv_nsec)))
```

Definition at line 62 of file timespec.h.

#define timespec_lt(time1, time2)

Value:

```
(((time1).tv_sec < (time2).tv_sec) || \
         (((time1).tv_sec == (time2).tv_sec) && \
         ((time1).tv_nsec  < (time2).tv_nsec)))
```

Definition at line 67 of file timespec.h.

#define timespec_ne(time1, time2)

Value:

```
(((time1).tv_sec != (time2).tv_sec) || \
         ((time1).tv_nsec != (time2).tv_nsec))
```

Definition at line 82 of file timespec.h.

#define timespec_set(time, newtime)

Value:

```
do { \
        (time).tv_sec = (newtime).tv_sec; \
        (time).tv_nsec = (newtime).tv_nsec; \
} while (0)
```

Definition at line 42 of file timespec.h.

#define timespec_sub(result, subtrahend)

Value:

```
do { \
```

```
        if ((result).tv_nsec >= (subtrahend).tv_nsec) { \
                (result).tv_nsec -= (subtrahend).tv_nsec; \
                (result).tv_sec -= (subtrahend).tv_sec; \
        } else { \
                (result).tv_nsec += NANOSEC_PER_SEC; \
                (result).tv_nsec -= (subtrahend).tv_nsec; \
                (result).tv_sec -= (subtrahend).tv_sec + 1; \
        } \
} while (0)
```

Definition at line 31 of file timespec.h.

#define timespec_valid(time)

Value:

```
((time).tv_sec >= 0 && \
         (time).tv_nsec >= 0 && \
        (time).tv_nsec <= NANOSEC_PER_SEC)
```

Definition at line 89 of file timespec.h.

urgency.c File Reference

Setup an initial resource set from Linux Resource Kernel.

```
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>
#include "ffd.h"
#include "urgency.h"
#include "event_trace.h"
```

Functions

int **et_chg_cpu_reserves** (int res, int per)

Allows for changes to cpu reservations This function returns an int upon completion: 0=success, -1=unsuccessful.

void **set_rk_thread_urgency** (enum ffd_task_type whichthread)

Sets up the urgency of specific threads of the global resource set within the resource kernel. This function returns a void type upon completion.

void **set_process_urgency** (const char *infop)

Establishes urgency of processes for Fast Failure Detection program. This function returns a void type upon completion.

void **set_thread_urgency** (enum ffd_task_type whichthread)

Sets up the urgency of specific threads of the global resource set within the resource kernel. This function returns a void type upon completion.

Detailed Description

Setup an initial resource set from Linux Resource Kernel.

Establishes the resource sets from the Linux Resource Kernel for the Fast Failure Detection program.

-*- Mode: C -*- **urgency.c** --- Author : Doug Wells Last Modified By : John Drummond Last Modified On : 2/24/02 Last Machine Used: quorum4 Update Count : 6

Definition in file **urgency.c**.

Function Documentation

int et_chg_cpu_reserves (int *res*, int *per*)

Allows for changes to cpu reservations This function returns an int upon completion: 0=success, -1=unsuccessful.

/*!

Definition at line 99 of file urgency.c.

References et_ffd, and remove_rk_resource_set().

```
100 {
101 //#define NS_PER_MS       1000000
102
103 cpu_reserve_attr_data_t attr;
104 int mfactor = 1000000;
105
106         int et_ret, i;
107
108         #ifdef EVENT_TRACE_DEBUG3
109         printf("Change CPU Reserves\n");
110         #endif
111
112         if(res != -1) {
113
114
115
116                 attr.compute_time.tv_nsec = res * mfactor;
117
118         }
119
120         if(per != -1) {
121
122
123
124
125                 attr.period.tv_nsec = per * mfactor;
126                 attr.deadline.tv_nsec = per * mfactor;
127
128
129
130         }
131
132
133
134         et_ret = rk_cpu_reserve_ctl(ffd_rs, &attr);
135         return(et_ret);
136 };
```

void set_process_urgency (const char * *infop*)

Establishes urgency of processes for Fast Failure Detection program. This function returns a void type upon completion.

Definition at line 818 of file urgency.c.

References create_rk_resource_set(), event_trace_struct::et_event_loc, event_trace_struct::et_event_task, event_trace_struct::et_event_type, et_ffd, event_trace_struct::et_path_len, and event_trace_record().

```
819 {
820
821 /*JD*/
822 #ifdef EVENT_TRACE
823
824
```

```
825
826 int et_rec_ret = 0;
827 #endif /* EVENT_TRACE */
828
829         #ifdef EVENT_TRACE_DEBUG3
830         printf("Set_process_urgency\n");
831         #endif
832
833         (void) infop;
834
835 #if     USE_RK_RESOURCE_SET
836         if (verbose)
837                 printf ("JD>>>URGENCY.C Using Linux/RK resource sets.\n");
838
839
840 /*JD QOS set here*/
841 #ifdef EVENT_TRACE
842 ++et_ffd->et_path_len;
843
844 et_ffd->et_event_type = PATH_LN;
845 et_ffd->et_event_loc = SET_PROC_URGENCY;
846 ++et_ffd->et_event_task.et_trace_lev;
847         if ((et_rec_ret = event_trace_record(et_ffd))!=0)
848         fprintf (stderr, "EVENT_TRACE
ERROR:URGENCY.C_SET_PROCESS_URGENCY_event_trace_record return %d\n",et_rec_ret);
849 #endif /* EVENT_TRACE */
850
851         ffd_rs = create_rk_resource_set ("ffd");
852 #else
853         if (verbose)
854                 printf ("No urgency applied.\n");
855                 /* do nothing   */
856 #endif  /* USE_RK_RESOURCE_SET  */
857
858         ++using_urgency;
859 }
```

void set_rk_thread_urgency (enum ffd_task_type *whichthread*)

Sets up the urgency of specific threads of the global resource set within the resource kernel. This function returns a void type upon completion.

Definition at line 736 of file urgency.c.

References et_sys_res::cpu_avail, et_sys_res::cpu_total, et_sys_res::cpu_used, event_trace_struct::et_event_task, et_ffd, event_trace_struct::et_path_len, event_trace_struct::et_res_level, event_trace_record(), et_sys_res::percent_cpu, and set_rk_resource_set_urgency().

```
737 {
738
739 /*JD*/
740 int et_rec_ret = 0;
741
742       #ifdef EVENT_TRACE_DEBUG3
743         printf("Set_rk_thread_urgency\n");
744       #endif
745
746       #ifdef EVENT_TRACE_DEBUG2
747       printf("Set_rk_thread_urgency\n");
748         switch (whichthread) {
749         case FFDTASK_HBEAT:
```

```
750                printf("Set_rk_thread_urgency WHICHTHREAD= FFDTASK_HBEAT -
THREAD2\n");
751
752
753
754         break;
755         case FFDTASK_CENSUS:
756                printf("Set_rk_thread_urgency WHICHTHREAD= FFDTASK_CENSUST -
THREAD1\n");
757
758
759
760         break;
761         case FFDTASK_MPLEX:
762                printf("Set_rk_thread_urgency WHICHTHREAD= FFDTASK_MPLEX\n");
763         break;
764         case FFDTASK_REPORT:
765                printf("Set_rk_thread_urgency WHICHTHREAD= FFDTASK_REPORT -
THREAD3\n");
766
767
768         break;
769         default:
770                printf("Set_rk_thread_urgency WHICHTHREAD= FFDTASK_REPORT -
DEFAULT\n");
771         break;
772         }
773       #endif
774
775
776 #if     USE_RK_RESOURCE_SET
777         switch (whichthread)
778         {
779         case FFDTASK_HBEAT:
780         case FFDTASK_CENSUS:
781         case FFDTASK_MPLEX:
782                         /* Only the heartbeat and censustakers have     */
783                         /* urgency needs that we honor here:            */
784
785         ++et_ffd->et_path_len;
786         ++et_ffd->et_event_task.et_trace_lev;
787         if ((et_rec_ret = event_trace_record(et_ffd))!=0)
788
fprintf(stderr,"ERROR:URGENCY.C_SET_RK_THREAD_URGENCY_event_trace_record return
%d\n",et_rec_ret);
789
790
791                 set_rk_resource_set_urgency (ffd_rs);
792                 break;
793         case FFDTASK_REPORT:
794         case FFDTASK_OTHER:
795                         /* These are valid threads that don't get any   */
796                         /* special urgency:                             */
797                 break;
798         default:
799         case FFDTASK_NONE:
800                         /* We don't think that these threads ought to   */
801                         /* exist:                                       */
802                assert (! "Unknown thread in set_rk_thread_urgency");
803         }
804
805 #endif  /* USE_RK_RESOURCE_SET  */
806
807
808
809         (void) whichthread;
810 }
```

void set_thread_urgency (enum ffd_task_type *whichthread*)

Sets up the urgency of specific threads of the global resource set within the resource kernel. This function returns a void type upon completion.

/*!

Definition at line 867 of file urgency.c.

References event_trace_struct::et_event_task, et_ffd, event_trace_struct::et_path_len, event_trace_record(), and set_rk_thread_urgency().

```
868 {
869
870 int et_rec_ret = 0;
871
872         #ifdef EVENT_TRACE_DEBUG3
873         printf("Set_thread_thread_urgency\n");
874         #endif
875
876         if (! using_urgency)
877                 return;
878
879
880
881
882 /*JD*/
883 ++et_ffd->et_path_len;
884 //et_ffd->et_event_loc = 6; /* loc 6 = THREAD2 */
885 ++et_ffd->et_event_task.et_trace_lev;
886 if ((et_rec_ret = event_trace_record(et_ffd))!=0)
887        fprintf (stderr, "EVENT_TRACE
ERROR:URGENCY.C_SET_THREAD_URGENCY_event_trace_record return %d\n",et_rec_ret);
888
889
890
891         set_rk_thread_urgency (whichthread);
892 }
```

urgency.h File Reference

Include file for resource set from Linux Resource Kernel.

Functions

void **set_process_urgency** (const char *infop)

Establishes urgency of processes for Fast Failure Detection program. This function returns a void type upon completion.

void **set_thread_urgency** (enum ffd_task_type whichthread)

Sets up the urgency of specific threads of the global resource set within the resource kernel. This function returns a void type upon completion.

int **et_chg_cpu_reserves** (int res, int per)

Allows for changes to cpu reservations This function returns an int upon completion: 0=success, -1=unsuccessful.

void remove_rk_resource_set (void)

Sets up the removal/destruction of the global resource set within the resource kernel. This function returns a void type upon completion.

rk_resource_set_t **create_rk_resource_set** (char *resource_set_name)

Sets up the creation of the global resource set within the resource kernel. This function returns a rk_resource_set_t type upon completion.

void **set_rk_resource_set_urgency** (rk_resource_set_t rs)

Sets up the urgency of the previously created global resource set within the resource kernel. This function returns a void type upon completion.

void **set_rk_thread_urgency** (enum ffd_task_type whichthread)

Sets up the urgency of specific threads of the global resource set within the resource kernel. This function returns a void type upon completion.

Variables

const char * **_urgency_h_file_version_G_** = "$Id: urgency.h,v 1.4 2001/08/16 18:16:02 mrahman Exp $"

Detailed Description

Include file for resource set from Linux Resource Kernel.

-*- Mode: C -*- **messages.h** --- Author : Doug Wells Last Modified By : Mustafizur Rahman Last Modified On : Thu Aug 16 14:08:20 2001 Last Machine Used: quite01.camb.opengroup.org Update Count : 13

Definition in file **urgency.h**.

Function Documentation

rk_resource_set_t create_rk_resource_set (char * *resource_set_name*)

Sets up the creation of the global resource set within the resource kernel. This function returns a rk_resource_set_t type upon completion.

/*!

Referenced by set_process_urgency().void remove_rk_resource_set (void)

Sets up the removal/destruction of the global resource set within the resource kernel. This function returns a void type upon completion.

/*!

Referenced by et_chg_cpu_reserves().void set_rk_resource_set_urgency (rk_resource_set_t *rs*)

Sets up the urgency of the previously created global resource set within the resource kernel. This function returns a void type upon completion.

/*!

Referenced by set_rk_thread_urgency().

Variable Documentation

const char* _urgency_h_file_version_G_ = "$Id: urgency.h,v 1.4 2001/08/16 18:16:02 mrahman Exp $" `[static]`

File version for concurrent version system program

Definition at line 45 of file urgency.h.

utils.c File Reference

Utilities function file for FFD exeriment.

```
#include <stdlib.h>
#include <stdio.h>
#include <stdarg.h>
#include <errno.h>
#include <string.h>
#include <assert.h>
#include <math.h>
#include "ffd.h"
```

Detailed Description

Utilities function file for FFD exeriment.

-*- Mode: C -*- **utils.c** --- Author : Doug Wells Last Modified By : Mustafizur Rahman Last Modified On : Thu Sep 27 12:32:23 2001 Last Machine Used: quite01.camb.opengroup.org Update Count : 71

utils - this module contains miscellaneous routines that are used in several other places but don't seem to deserve a separate module. -Doug Wells

Definition in file **utils.c**.

utils.h File Reference

Include file for utilities function in FFD exeriment.

```
#include <pthread.h>
```

Detailed Description

Include file for utilities function in FFD exeriment.

-*- Mode: C -*- **utils.h** --- Author : Doug Wells Last Modified By : Mustafizur Rahman Last Modified On : Fri Aug 31 17:09:04 2001 Last Machine Used: quite01.camb.opengroup.org Update Count : 39

utils.h - include file to declare utility functions. --Doug Wells

Definition in file **utils.h**.

C. COPYRIGHT AND LICENSE

This section provides the copyright and license notices for all application program source code files, operating system and related system source code files, as well as specific kernel module systems and source code files. All these elements have been applied towards this dissertation research effort.

1. Fast Failure Detector

This notice references the Fast Failure Detector software program files that employ the Linux/RK resource kernel system and has been likewise utilized in this dissertation research effort. The source code for the fast failure detection program was developed under the DARPA Quorum Integration program.

2. Linux/RK

This notice references the Linux/RK resource kernel system and any/all files that have been utilized in this dissertation research effort. The sources code for the linux resource kernel system was developed under the DARPA Quorum Integration program.

Carnegie Mellon requests users of this software to return to Real-Time and Multimedia Systems Laboratory Attn: Prof. Raj Rajkumar Electrical and Computer Engineering, and Computer Science Carnegie Mellon University Pittsburgh PA 15213-3890 or via email to `raj@ece.cmu.edu` any improvements or extensions that they make and grant Carnegie Mellon the rights to redistribute these changes.

3. Linux

This notice references the operating system that was utilized in the dissertation research effort.

Linux is written and distributed under the GNU General Public License which means that its source code is freely-distributed and available to the general public.

GNU GENERAL PUBLIC LICENSE Version 2, June 1991, Copyright (C) 1989, 1991 Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA, Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under

copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may

redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it. Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute

or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not

thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found. <one line to give the program's name and a brief idea of what it does.> Copyright (C) 19yy <name of author>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode: Gnomovision version 69, Copyright (C) 19yy name of author Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'. This is free software, and you are welcome to redistribute it under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w' and `show c'; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names: Yoyodyne, Inc., hereby disclaims all copyright interest in the program `Gnomovision' (which makes passes at compilers) written by James Hacker.

<signature of Ty Coon>, 1 April 1989 Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

4. Linux System Administrator Guide Figure

The Linux System Administrator's Guide: Version 0.7

Copyright 2001 Stephen Stafford, Copyright 1998--2001 Joanna Oja, Copyright 1993--1998 Lars Wirzenius.

Trademarks are owned by their owners.

5. Ensemble

This software was developed by members of the Horus Project:

Staff:

Robbert van Renesse: (607) 255-1021; FAX: (607) 255-4428; e-mail: rvr@cs.cornell.edu

Kenneth P. Birman: (607) 255-9199; FAX: (607) 255-4428; e-mail: ken@cs.cornell.edu

Werner Vogels: (607) 255-9196; FAX: (607) 255-4428; e-mail: vogels@cs.cornell.edu

Tim Clark: (607) 255-4117; FAX: (607) 255-4428; e-mail: tclark@cs.cornell.edu

Students:

Mark Hayden: mh37@cornell.edu

Alexey Vaysburd:

Department of Computer Science

Cornell University

Ithaca, New York 14853

For more information contact one of us directly.

LICENSE TERMS AND CONDITIONS

1. The 'Software', below, refers to the Ensemble system, developed by the Horus Project (in either source-code, object-code or executable-code form), and related documentation, and a 'work based on the Software' means a work based on either the Software, on part of the Software, or on any derivative work of the Software under copyright law: that is, a work containing all or a portion of the Ensemble System, either verbatim or with modifications. Each licensee is addressed as 'you' or 'Licensee.'

2. Cornell University as the parent organization of the Horus Project holds copyrights in the Software. The copyright holder reserves all rights except those expressly granted to licensees, and U.S. Government license rights.

3. Permission is hereby granted to use, copy, modify, and to redistribute to others. If you distribute a copy or copies of the Software, or you modify a copy or copies of the Software or any portion of it, thus forming a work based on the Software, and make and/or distribute copies of such work, you must meet the following conditions:

a) If you make a copy of the Software (modified or verbatim) it must include the copyright notice and this license.

b) You must cause the modified Software to carry prominent notices stating that you changed specified portions of the Software.

4. LICENSEE AGREES THAT THE EXPORT OF GOODS AND/OR TECHNICAL DATA FROM THE UNITED STATES MAY REQUIRE SOME FORM OF EXPORT CONTROL LICENSE FROM THE U.S. GOVERNMENT AND THAT FAILURE TO OBTAIN SUCH EXPORT CONTROL LICENSE MAY RESULT IN CRIMINAL LIABILITY UNDER U.S. LAWS.

5. Portions of the Software resulted from work developed under a U.S. Government Contract and are subject to the following license: the Government is granted for itself and others acting on its behalf a paid-up, nonexclusive, irrevocable worldwide license in this computer software to reproduce, prepare derivative works, and perform publicly and display publicly.

6. Disclaimer of warranty: Licensor provides the software on an ``as is'' basis. Licensor does not warrent, guarantee, or make any representations regarding the use or results of the software with respect to its correctness, accuracy, reliability or performance. The entire risk of the use and performance of the software is assumed by licensee.

ALL WARANTIES INCLUDING, WITHOUT LIMITATION, ANY WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR MERCHANTABILITY ARE HEREBY EXCLUDED.

7. Lack of maintainance or support services: Licensee understands and agrees that licensor is under no obligation to provide maintanance, support or update services, notices of latent defects, or correction of defects for the software.

8. Limitation of liability, indemnification: Even if advised of the possibility of damages, under no circumstances shall licensor be liable to licensee or any third party for damages of any character, including, without limitation, direct, indirect, incidental, consequential or special damages, loss of profits, loss of use, loss of goodwill, computer

failure or malfunction. Licensee agrees to indemnify and hold harmless licensor for any and all liability licensor may incur as a result of licensee's use of the software.

DEFINITIONS, SYMBOLS, AND ACRONYMS

AEGIS

Advanced Electronic Guidance and Instrumentation System, A totally integrated shipboard weapon system that combines computers, radars, and missiles to provide a defense umbrella for surface shipping.

QoS

Quality of Service, The specific measures of provisioning shared resources in a logical way that can be based upon the priorities, needs, and/or requirements of applications.

C2

Command & Control, The facilities, equipment, communications, procedures, and personnel essential to a commander for planning, directing, and controlling operations of assigned forces pursuant to the missions assigned.

C4

Command, Control, Communication, Computers, The integrated systems of doctrine, procedures, organizational structures, personnel, equipment, facilities, and communications designed to support a commander's exercise of command and control across the range of military operations.

Distributed Processing

Any environments that perform computing functions are stated to be distributed if the application software structure and data computations within these environments are dispersed over two or more computing entities. These distributed processing computers will communicate by means of some form of interconnected network.

Event

A behavioral performance related action.

Kernel

An operating system element which seperates hardware resources from the application programs and users.

QMS

Quality of Service Metric Service, experiment control software(developed through DARPA Quorum) to be utilized during event trace research experiment which, displays the configuration and network connectivity of the testbed, provides a GUI to perform general experiment control of the testbed, displays a view of all active processes on each node, and provides the ability to request, obtain and display general system metrics (CPU load, memory usage, disk I/O)

Command & Control Path

A collection of modules which are serially accessed within the facilities, equipment, communications, procedures, and personnel utilized by a commander for planning, directing, and controlling operations of assigned forces pursuant to the missions assigned. These modules include categories of Sensor, Filter, Evaluate/Decide, and Actuator.

Situation Assessment

Assessment produced by combining military geography, weather, and threat data to provide a comprehensive projection of the situation for the decisionmaker.

Strategic (level)

Activities at this level establish national and multinational military objectives; sequence initiatives; define limits and assess risks for the use of military and other instruments of national power; develop global plans to achieve these objectives; and provide military forces and other capabilities in accordance with strategic plans. Strategic operations are designed to have a long-range, rather than immediate, effect on the enemy and its military forces.

Tactical (level)

Activities at this level focus on the ordered arrangement and maneuver of combat elements in relation to each other and to the enemy to achieve combat objectives.

Operational (level)

Activities at this level link tactics and strategy by establishing operational objectives needed to accomplish the strategic objectives, sequencing events to achieve the operational objectives, initiating actions, and applying resources to bring about and sustain these events. These activities imply a broader dimension of time or space than do tactics; they ensure the logistic and administrative support of tactical forces, and provide the means by which tactical successes are exploited to achieve strategic objectives.

QoS Execution Pathway

The consecutive execution of quality of service and/or resource management specific statements within a system.

LIST OF REFERENCES

[Adriole 86] Adriole, Stephen, J., *Software Validation, Verification and Testing Technique and Tool Reference Guide*, Petrocelli Books Inc., Princeton, New Jersey, 1986.

[Auguston 00] Auguston, M., *Assertion Checker For The C Programming Language Based On Computations Over Event Traces*, Fourth International Workshop on Automated Debugging, AADEBUG2000, Munich, Germany, August 2000.

[Auguston 99] Auguston, M., *Tools For Program Dynamic Analysis, Testing, And Debugging Based Upon Event Grammars*, Technical Report, NMSU CSTR-9906, Department of Computer Science, New Mexico State University, New Mexico, June 1999.

[Auguston 98] Auguston, M., *Building Program Behavior Models*, Proceedings of the European Conference on Artificial Intelligence ECAI-98, Workshop on Spatial and Temporal Reasoning, Brighton, England, August 23-28, 1998.

[Auguston 92] Auguston, M., *PARFORMAN-an Assertion Language for Specifying behaviour when Debugging Parallel Applications*, Technical Report, LiTH-IDA-R-92-16, Department of Computer and Informantion Science, Linkoping University, Sewden, 1992.

[Barta 95] Barta, R., *Formal Specification of Distributed Systems A Discrete Space-Time Logic*, Ph.D. Dissertation, Technischen Universitat Wien, Technisch-Naturwissennschaftliche Fakultat, Vienna April 1995.

[Berzins 91] V. Berzins and Luqi, *Software Engineering with Abstractions*, Addison-Wesley 1991.

[Berzins 99] V.Berzins, Luqi, M.Shing, M.Saluto, J.Williams, *Re-engineering the Janus(A) Combat Simulation System*, Naval Postgraduate School, Technical Report, NPS-CS-99-004. 1999.

[Birman 97] Birman, K., Vogels, W., Guo, K., Hayden, M., Hickey, T., Friedman, R., Maffeis, S., VanRenesse, R., Vaysburd, A., "*Moving the Ensemble Groupware System to Windows NT and Wolfpack*", Proceedings of USENIX Windows NT Workshop, Seattle, WA. August 11-13, 1997.

[Birman 00] Birman, K., et al., *"The Horus and Ensemble Projects: Accomplishments and Limitations"*, Proceedings of the DARPA Information Survivability Conference & Exposition (DISCEX '00), Hilton Head, South Carolina, January 2000.

[Blair 98] Blair, L., Blair, G., *The Impact of Aspect-Oriented Programming on Formal Methods*, Computing Department, Lancaster University, Bailrigg, Lancaster, LA1 4YR, Internal Report No: MPG-98-08, May 1998.

[Boyes 97] Boyes J. L. & Andriole S. (eds.), *Principles of Command and Control*, Washington: AFCEA International Press, 1987.

[Cheung 98] K.S. Cheung, K.O. Chow and T.Y. Cheung *Deriving Scenarios Of Object Interaction Through Petri Net,* Proceedings Of The Technology Of Object-Oriented Languages And Systems, 1998.

[Cicalese 99] Cicalese, C., Rotenstreich, S., *Behavioral Specification of Distributed Software Component Interfaces,* Computer, Vol 32, No. 7, pp. 46-53 July, 1999

[Cisco 00] *Introduction: Quality of Service Overview,* Quality of Service and architectures, Cisco Systems Inc. Def. May, 2000.

[CMU 97] Carnegie Mellon University, *What is a Resource Kernel,* Computer Science Department, Real-Time and Multimedia Laboratory Web-site, http://www-2.cs.cmu.edu/afs/cs/project/art-6/www/resource-kernel.html, October, 14, 1997

[Cooper 94] Cooper, C., *Complexity in C3I systems,* Complexity International, Volume 1, Paper ID: cooper01, 1994.

[DOD 00] Ed., *DOD Dictionary of Military and Associated Terms,* Joint Publication 1-02, Joint Doctrine Division, J-7, Joint Staff, July 2000.

[Drummond 02] Drummond, J., Wells, D., Rahman, M., *Detecting Failure Within Distributed Environments,* SPAWAR Technical Report TR1884, Space and Naval Warfare Systems Center, San Diego, Ca. 2002.

[Flores 99] Flores, S. Levine, D. Schmidt, D. *An Empirical Evaluation of OS Support for Real-time CORBA Object Request Brokers(ORB).* The Real-Time

Technology and Applications Symposium (RTAS), Vancouver, British Columbia, Canada, June 2, 1999.

[Foldoc 01] *Free Online Dictionary Of Computing*, Imperial Collage Department of Computing, United Kingdom, 2001.

[Frolund 98] Frolund, S., Jari Koistinen, J., *Quality of Service (QoS) Specification in Distributed Object Systems*, Hewlett-Packard Laboratories, Proceedings of the 4th USENIX Conference on Object-Oriented Technologies and Systems (COOTS), Santa Fe, New Mexico, April 27-30, 1998

[Fu 99] Fu, T., *A Design and Implementation of Dynamic Real-time Benchmark for Assessment of QoS and Resource Management Technology*, Master Thesis, Department of Computer Science Engineering, University of Texas, May, 1999.

[Guerin 98] Roch Guerin, IBM Research *Keynote Address*, 1998 Sixth International Workshop on Quality of Service, Napa, California, May, 1998.

[Harel 97] David Harel, Eran Gery, *Executable Object Modeling with Statecharts*, IEEE Computer, Proceedings, 18th International Conference on Software Engineering, IEEE Press, March 1996, Revised February, 1997.

[Hariri 95] Salim Hariri, Hasan Mutlu, *Hierarchical Modeling of Availability in Distributed Systems*, IEEE Transactions on Software Engineering, Vol 21, No. 1, Jan 1995.

[Harn 99] Harn, M. Berzins, V. Luqi, Kemple, W., *Evolution of C4I Systems*, Command & Control Research and Technology Symposium, 1999.

[Hrischuk 99] Curtis E.Hrischuk, C.Murray Woodside, Jerome A. Rolia, Rod Iversen, *Trace-Based Load Characterization for Generating Performance Software Models*, IEEE Transactions On Software Engineering, Vol. 25, No. 1, January/February 1999

[IDEF1X 93] *Integration Definition For Information Modeling (Idef1x)*, Federal Information Processing Standards Publication 184, National Institute of Standards and Technology, December 21, 1993.

[Irvine 00] Cynthia Irvine, Timothy Levin, *Quality of Security Service*, Proceedings of the New Security Paradigms Workshop, Cork, Ireland, September 2000.

[Johnson 99] Johnson, V. & M. *Higher Level Protocols used with IP Multicast*, 1999.

[Komarinski 00] Kmoarinski, M., Collett, C. *Red Hat Linux System Administration Handbook*. Prentice Hall, Inc. Upper Saddle River, New Jersey. 2000.

[Koob 99] G.Koob, *Background for DARPA Quorum Mission Statement.* Defense Advanced Research Projects Agency, Information Technology Office. 1999.

[Kopetz 97] Hermann Kopez, *Real-Time Systems Design Principles for Distributed Embedded Applications*, Kluwer Academic Publishers, 1997.

[Kornegay 99] Kevin T. Kornegay, Gang Qu, Miodrag Potkonjak, *Quality of Service and System Design* ,Proceedings of the IEEE Computer Society Workshop on VLSI'99 1999.

[Lee 99] Lee, C., *On Quality of Service Management*, Ph.D. Dissertation, Department of Computer Science, Carnegie Mellon University, August, 1999.

[Lounsbury 99] Lounsbury, D., *DARPA Quorum Integration Review,* July 1999.

[Luqi 88] Luqi, M.Ketabehi, *A Computer Aided Prototyping System*, IEEE Software 5 (2) 66-72,

March 1988.

[Luqi O88] Luqi, V.Berzins, R.Yeh, *A Prototyping Language for Real-Time Software,* IEEE Transaction on Software Engineering 14(10) 1409-1423, October 1988.

[Maher 96] Maher, A., et. al., *Object-Oriented Technology for Real-Time Systems: A Practical Approach Using OMT and Fusion,* Prentice-Hall, Inc., Simon & Schuster Co., New Jersey, 1996.

[Marciniak 94] Ed. John J. Marciniak, *Encyclopedia of Software Engineering*, Vol. 2, pp 978-984" John Wiley & Sons. ISBN: 0471540048, 1994.

[Matsui 98] Yasunori Matsui, Seiji Kihara, Atsushi Mitsuzawa, Satoshi Moriai, Hideyuki Tokuda, *An Extensible Object Model for QoS Specification in Adaptive QoS Systems,* Proceedings of the Second IEEE International Symposium on Object-Oriented Real-Time Distributed Computing 1998.

[Meyer 97] Meyer, B., *Object-Oriented Software Construction,* 2nd ed., Prentice Hall, Upper Saddle River, J.J., 1997.

[MIRnet 99] *Definitions of Quality of Service,* MIRnet consortium: University of Tennessee, Knoxville, Russian Institute for Public Networks (RIPN), Moscow State University (MSU), Friends and Partners-Russia (F&P/R), the Russian Academy of Science, VUZTelecom Center of St. Petersburg, July, 1999.

[Montiel 93] Montiel, J. et. al., *Methods for QoS Verification and Protocol Conformance Testing in IBC –Application Guidelines*, Technical Report R2088, Dassault Automatismes & Telecommunications, Etnoteam S.P.A., Gesellschaft fur Mathematik und Datenverarbeitung FOKUS Berlin, Intracom S.A., Siemens A.G., Technische Universitat Berlin, University of Sterling, Verilog S.A., June 1993.

[Mosberger 97] David Mosberger, *Scout: A Path-Based Operating System*, Ph.D. Thesis, Department of Computer Science, Graduate College, University Of Arizona 1997.

[Oikawa 98] Oikawa, S., Rajkumar, R., *Linux/RK: A Portable Resource Kernel in Linux*, IEEE Real-Time Systems Symposium Work-In-Progress, Madrid, December 1998.

[Pryce 00] Nathaniel Pryce, *Component Interaction in Distributed Systems* ,Ph.D. Thesis, Department of Engineering of the University of London, and for the Diploma of the Imperial College of Science, Technology and Medicine January 2000.

[Rajkumar 97] Rajkumar, R., Lee, C., Lehoczky, J., Siewiorek, D., *A resource allocation model for QoS management*, Proceedings of The 18th IEEE Real-Time Systems Symposium, pp298-307. 1997.

[Rajkumar 98] Rajkumar, R., Juvva, K., Molano, A., Oikawa, S., *Resource Kernels: A Resource-Centric Approach to Real-Time and Multimedia*, In Proceedings of the SPIE/ACM Conference on Multimedia Computing and Networking, January 1998.

[Rajkumar 00] Rajkumar, R., *The Resource Kernel Project Linux/RK*, DARPA Quorum PI Meeging Orlando, Fl, December 2000.

[Rusling 99] Rusling, David, A., *The Linux Kernel*, Version 0.8-3, http://www.linuxdoc.org/LDP/tlk/tlk.html, 1996-1999.

[Rumbaugh 91] Rumbaugh, J., et. al., *Object Oriented Modeling and Design*, Prentice-Hall, Inc., Simon & Schuster Co., New Jersey, 1991.

[Sabata 97] Bikash Sabata, Saurav Chatterjee, Michael Davis, Jaroslaw Sydir, Thomas Lawrence, *Taxonomy for QoS Specifications*, Proceedings of Workshop on Object-oriented Real-time Dependable Systems (WORDS 97), February 1997.

[Sariowan 95] H. Sariowan, R.L. Cruz, G.C. Polyzos. *Scheduling for quality of service guarantees via service curves*. Proceedings Fourth International Conference on Computer Communications and Networks (ICCCN'95), pp. 512-520, 1995.

[Staehli 95] Staehli, R., *Quality of Service Specification for Resource Management in Multimedia Systems*, Ph.D. Dissertation, Oregon Graduate Institute, Oregon 1995.

[Stafford 01] Stafford, S., Wirzenius, L., Oja, J., *The Linux System Administrator's Guide, Version 0.7*, http://www.linuxdoc.org/LDP/sag/, 2001.

[TimeSys 00] TimeSys Corporation, *TimeSys Linux/RT Version 1.0 User's Manual*, Pittsburgh, Pennsylvania, 2000.

[Web 00] Wepedia, *Online Encylopedia of Computer Technology*, Copyright 1999-2000 internet.com Corp. July 2000.

[Welch 99a] Welch, L., Shirazi, B., *A Dynamic Real-time Benchmark for Assessment of QoS and Resource Management Technology*, Proceedings of the IEEE Real-time Technology and Applications Symposium, 36-45, June 1999.

[Welch 97] Welch,L. *DynBench: A Dynamic Real-Time Benchmark Suite and environment*, 1997

[Welch 98] Welch,L. *DeSiDeRaTa: Resource and QoS Management for Dynamic, Scalable, Dependable Real-Time Systems*, 1998.

[Welch 99] Welch,L. *Specification, Modeling and Analysis of Dynamic, dependable Real-Time Systems*, 1999.

[Xie 98] Xie, G., *SAAM: Integrated Network Architecture for Integrated Services Server and Agent Based Active Network Management Architecture,* 1998 Sixth International Workshop on Quality of Service, Napa, California, May, 1998.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center — 1
   Ft. Belvoir, Virginia

2. Dudley Knox Library — 2
   Naval Postgraduate School
   Monterey, California

3. Center for Naval Analysis — 1
   4401 Ford Avenue
   Alexandria, Virginia 22302-0268

4. Chief of Naval Research — 1
   800 N. Qunicy Street
   Arlington, Virginia 22217

5. Dr. Valdis Berzins, Code CS/Be — 1
   Naval Postgraduate School
   Montery, California 93943-5100

6. Dr. Luqi, Code CS/Lq — 1
   Naval Postgraduate School
   Montery, California 93943-5100

7. Dr. Mikhail Auguston — 1
   Naval Postgraduate School
   Montery, California 93943-5100

8. Dr. Nabendu Chaki — 1
   Naval Postgraduate School
   Montery, California 93943-5100

9. Dr. William Kemple — 1
   Naval Postgraduate School
   Montery, California 93943-5100

10. Dr. Man-Tak Shing, Code CS/Sh — 1
    Naval Postgraduate School
    Montery, California 93943-5100

11. Library, Code 20274 — 2
    Space and Naval Warfare Systems Center, San Diego
    San Diego, California 92152-5001

12. Hon. John W.Douglas 1
Assistant Secretary of the Navy
(Inferences, Research, Development and Acquisition)
Room E741
1000 Navy Pentagon
Washington, DC 20350-1000

13. John Drummond, Scientist, Code 24121 2
Space and Naval Warfare, Systems Center
San Diego, California 92152-5001

14. Dr. Marvin Langston 1
1225 Jefferson Davis Highway
Crystal Gateway 2 / Suite 1500
Arlington, Virginia 22202-4311

15. National Science Foundation 1
Attn: Bill Agresty
4201 Willson Blvd.
Arlington, Virginia 22230